Diplomamunka

Szabó Zsolt

Debrecen
2007

Debreceni Egyetem
Informatikai Kar

Web alapú levelezőrendszer fejlesztése ingyenes eszközök felhasználásával

Témavezető:

Dr. Kuki Attila
egyetemi adjunktus

Készítette:

Szabó Zsolt
programtervező matematikus

Debrecen
2007

Tartalomjegyzék

1. Bevezetés

1.1 Diplomamunkám témája

Az e-mail (az angol electronic mail-ből származik) története egészen 1965-ig vezethető vissza, amikor még internet nem létezett. Az azóta eltelt közel fél évszázad alatt az internet rohamos fejlődésnek indult, és pillanatok alatt elterjedt. Ennek köszönhetően az elektronikus levelezést is egyre szélesebb körben használják világszerte.

Diplomamunkám célja egy olyan web alapú alkalmazás elkészítése Java nyelven, amely e-mail-ek kezelésére szolgál.

1.2 Témaválasztás indoklása

A mai világban rengeteg e-mail szerver üzemel, jobbnál jobb szolgáltatásokat kínálva a leendő felhasználóknak. Meggyőződésem, hogy többségben vannak azok a rendszeres e-mail használók, akiknek egyszerre több szolgáltatónál is van érvényes e-mail címük. Az alapgondolatom egy olyan egységes web-alapú e-mail-kezelő, amely be tud jelentkezni különböző szolgáltatók fiókjaiba, és kezelni tudja az ott tárolt e-mail-eket. Számos ilyen program létezik (pl. Outlook, Mozilla Thunderbird, stb.), hátrányuk viszont az, hogy ezeket általában egy bizonyos gépre konfiguráljuk, a kliens gépére. Célom tehát az volt, hogy az egységes e-mail-kezelés gépfüggetlen legyen, és ezt egy szerver végezze. A feladat jellegéből adódóan kizárólag webes technológia jöhetett szóba.

Azonban elsődleges célom ezzel a feladattal a gyakorlás, tapasztalatszerzés és a különböző Java technológiák mélyebb elsajátítása volt.

2. Felhasznált technológiák és eszközök

2.1 Java technológia és előnyei

A komponens alapú tervezés szemszögéből is az egyik legjobb választás a Java technológia. Egyrészről az egyes részproblémák megoldásához szükséges eszközöket különböző csomagokban találjuk, ami arra serkent, hogy a feladatot kisebb részekre, különböző rétegekre bontsuk (például a klasszikus MVC modell szerint). A Java másik nagy előnye a platformfüggetlenség. A fordítás után előálló bájtkód tetszőleges platformon ugyanazt az eredményt szolgáltatja, és a programozónak/fejlesztőnek nem kell az értékes

idejét operációs rendszer-szintű kérdésekkel töltenie. A Java egyik további előnye, hogy a nyelv és a hozzá készült IDE-k többsége ingyenes. Ez gyakran jelentős megtakarítást jelent. Az eddig elmondottak alapján akár a .NET keretrendszert is választhattam volna, mert az is hasonló tulajdonságokkal rendelkezik: hordozható IL kódra fordít, támogatja a komponens-alapú tervezést, és igen kiterjedt eszközkészlete van. Választásom azért esett mégis a Javára, mert nem gyártó-függő. Szemben a Microsoft-termékként nyilvántartott .NET-tel, a Javának különböző implementációi vannak forgalomban.

2.1.1 Servlet

Egy servlet olyan speciális Java program, amely szorosan együttműködik egy webszerverrel, így lehetővé teszi a szerveroldalon HTML oldalak dinamikus létrehozását és paraméterezését. A servletek kizárólag szerverfunkciókat képesek ellátni. Ugyanerre a célra használható nem Java alapú technológiák pl. PHP, CGI, ASP.NET.

Ahhoz, hogy egy servlet objektumot kapjunk, implementálnunk kell a `javax.servlet.Servlet` interfész 3 metódusát:

- `void init(ServletConfig config)` – inicializálás céljából akkor fut le, amikor példányosítottunk egy servletet.
- `void service(ServletRequest req, ServletResponse res)` – akkor fut le, ha a kliensoldalon hívás érkezik a servlet felé. Ekkor a servlet a `ServletRequest` kérésre a `ServletResponse` válasszal felel.
- `void destroy()` – akkor fut le, ha többé már nincs szükségünk a servletre.

2.1.2 JSP (Java Server Pages)

A JSP egy magasabb absztrakciós szintet képvisel, mint a servlet. Segítségével lehetőségünk nyílik arra, hogy egy HTML oldalra JSP tagek közé Java kódot szúrjunk be. Ha kérés érkezik a webszerver felé egy JSP oldal elérésére, akkor a fordító a JSP oldalból servletet generál, majd ezt lefordítja bájtkódra. A bájtkódot a webszerver futtatja, kimenete a beérkezett kérésre adott válasz lesz. Egy JSP oldal servletté alakítására az első kérés kiszolgálásakor kerül sor, ekkor lassabban töltődik be az oldal, de miután elindult a servlet, a további kérések már egyenesen a servlethez kerülnek.

Egy webalkalmazás fejlesztésekor célszerű ötvözni a servletes és a JSP technológiát, melyek közt az összeköttetést JavaBean-ekkel valósíthatjuk meg. Erre alapozta az Apache a Struts néven elterjedt technológiáját.

2.1.3 Struts

Az Apache által fejlesztett Struts olyan nyílt forráskódú keretrendszer, amely webes Java alkalmazások fejlesztését könnyíti meg. Megszületését az indokolta, hogy amíg a kizárólag JSP technológiát felhasználó webalkalmazásokban gyakran keverednek az adatbázis-kezeléssel, a megjelenítéssel (pl. HTML), illetve a vezérléssel kapcsolatos kódok, addig a Struts keretrendszert használó rendszerekben ez a három jól elkülönül. Teljes egészében megvalósítja az MVC (*modell-view-controller*) architektúra modellt, melyben a *modell* szón az adatbázis-kezeléssel járó kódokat, a *view* szón a megjelenítéssel járó kódokat, míg a *controller* szón a vezérléssel járó kódokat értjük.
Az ilyen rendszerek karbantartása, illetve fejlesztése is költségkímélőbb.
A Struts kapcsán két fontos dologról kell beszélünk:

- `StrutsActionForm`, melyek nem mások, mint egyszerű JavaBean-ek. Az a feladatuk, hogy egy weblap űrlapjairól érkező adatokat eltárolják megfelelő attribútumok segítségével. Ahhoz, hogy egy általunk megírt JavaBean `StrutsActionForm` legyen, az `org.apache.struts.action.ActionForm` osztályból kell leszármaztatnunk. Egy ilyen bean rendelkezik egy speciális metódussal:
  `public ActionErrors validate(ActionMapping mapping, HttpServletRequest request)`, mely egy űrlapról érkező adatok helyességét hivatott ellenőrizni. Visszatérési értéke egy olyan `ActionErrors` objektum, mely az esetleges hibaüzeneteket tárolja, például hogy az űrlapról érkező mely adatokkal nem voltunk megelégedve.

- `StrutsAction`, melyeknek feladata, hogy egy űrlapról érkező adatokat feldolgozzanak. Ahhoz, hogy egy ilyen objektumot kapjunk, az `org.apache.struts.action.Action` absztrakt osztályból kell leszármaztatnunk osztályunkat, valamint implementálnunk kell a

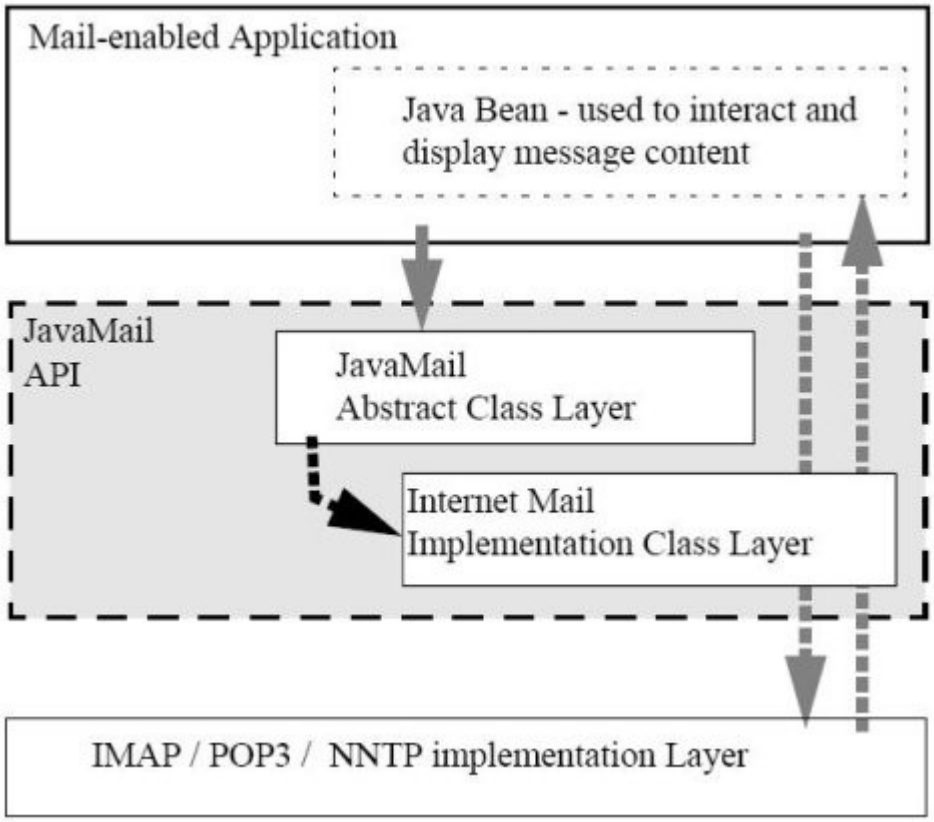
  ```
  public ActionForward execute(ActionMapping mapping, ActionForm  form,
        HttpServletRequest request, HttpServletResponse response)
  ```

  metódust, melynek második paramétere egy `StrutsActionForm` objektum.

A fent említett két eszköz legfontosabb tulajdonságai a `struts-config.xml` nevű fájlban találhatók meg. Ezt nevezhetjük a Struts technológia lelkének is, mivel itt vannak deklarálva, hogy például űrlapok elküldésekor, illetve linkre kattintáskor milyen `StrutsAction` eseménykezelő hajtódjon végre, és ehhez milyen `StrutsActionForm` bean tartozik. A WebMail alkalmazáshoz tartozó `struts-config.xml` egy részlete megtekinthető a függelékben.

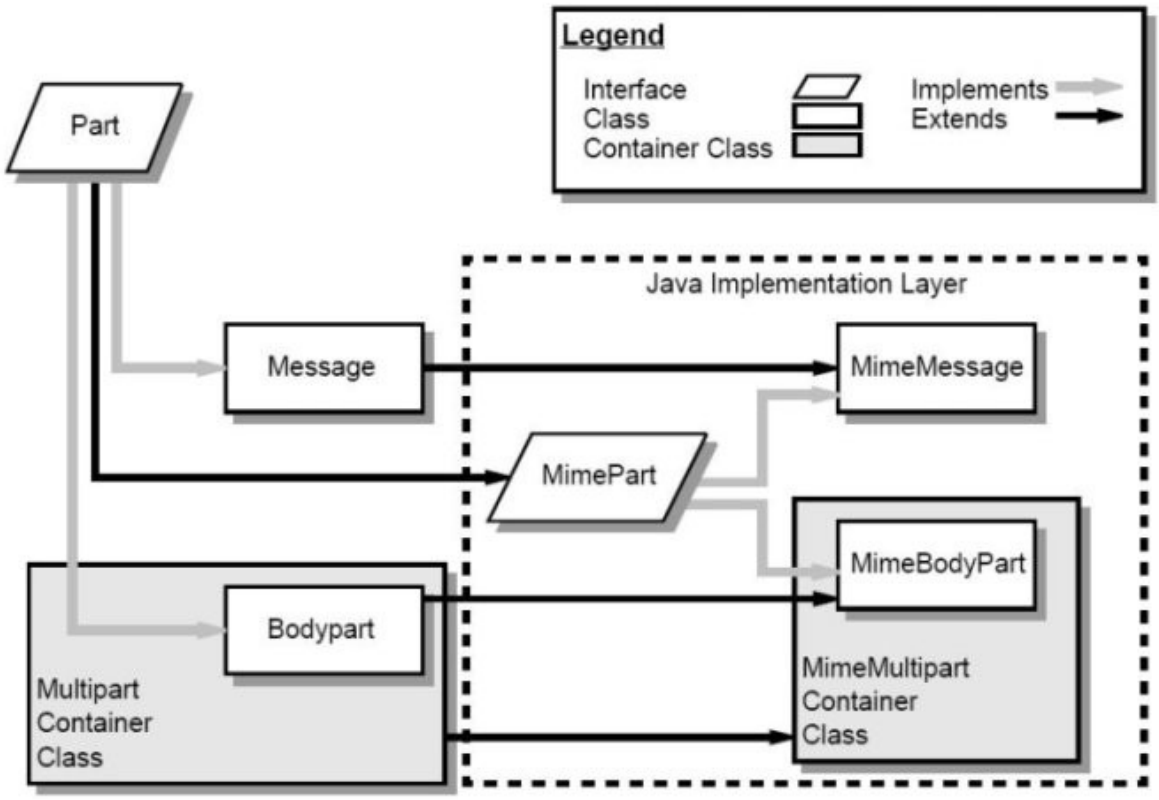
2.1.4 javax.mail API

Ahhoz, hogy egy általunk megírt program egy e-mail szerverrel kommunikálni tudjon, egy hálózati kapcsolatot kell felépítenünk, és ezek után az RFC 821 szabvány szerint rögzített parancsok használatával vehetjük igénybe egy ilyen szerver szolgáltatásait. Természetesen nem lehetetlen, de ezt eléggé körülményes kivitelezni, és nagy odafigyelést igényel a programozótól. Ezért is alkotta meg a Sun Microsystems a `javax.mail` API-t, amely nagy részben megkönnyíti az e-mailek feldolgozását Java nyelven. A `javax.mail` API az e-mail szerver és a kliens oldali email-kezelő alkalmazás között helyezkedik el, ezzel tulajdonképpen elrejtve előlünk az email szerverrel történő tényleges kommunikációt. (1. ábra)



1. ábra

A `javax.mail` API legfontosabb eleme a `javax.mail.Part` interfész. Ebben az interfészben vannak specifikálva olyan alapvető metódusok, amelyek egy e-mail kezelésénél elengedhetetlenek. Ezeknek egy részét implementálja a `javax.mail.Message` absztrakt osztály. A tényleges megvalósítás azonban a `javax.mail.internet.MimeMessage` osztály révén valósul meg, amely a `Message` absztrakt osztály leszármazottja. Egy ilyen objektum képvisel egy teljes e-mailt, annak összes információjával együtt. A `Part` interfészt ugyanakkor egy `javax.mail.BodyPart` osztály is implementálja, amelynek leszármazott osztálya a `javax.mail.internet.MimeBodyPart`. Összetett e-mailek esetén az egyes összetevők típusa `MimeBodyPart` lesz. Összetett e-mailekről akkor beszélünk, amikor szöveges üzenet mellett például csatolmány is érkezik. A következő ábra az egyes interfészek és osztályok hierarchiáját, illetve a köztük lévő kapcsolatokat hivatott bemutatni (2. ábra):



2. ábra

Logikailag szemlélve egy `Message` objektumot két részre bonthatunk (3. ábra):

- fejléc attribútumokra, és ezek kezelő metódusaira

  A fejlécek információt adnak egy e-mail tartalmáról, például milyen jellegű a benne tárolt szöveges információ, milyen karakterkódolást használtak, mikor küldték az e-mailt stb. Az egyik legfontosabb ilyen fejléc a '`Content-Type`' néven elérhető fejléc. Amint nevéből is adódik, az email tartalmi formájára utal, ezek közül is a legfontosabbak:

  - `'text/plain'` – sima szöveges tartalom.
  - `'text/html'` – a szöveg HTML-szerűen van formázva.
  - `'application/pdf'` – pdf kiterjesztésű csatolmány tartozik az e-mailhez.
  - `'multipart/alternative'` – összetett típusú e-mail, amely általában két részből áll. Az egyik `'text/plain'`, a másik `'text/html'` formátumban tartalmazza ugyanazt a szöveget.
  - `'multipart/mixed'` – összetett típusú e-mail, amely arra utal, hogy az összetevő komponensek bármilyen típusúak lehetnek.

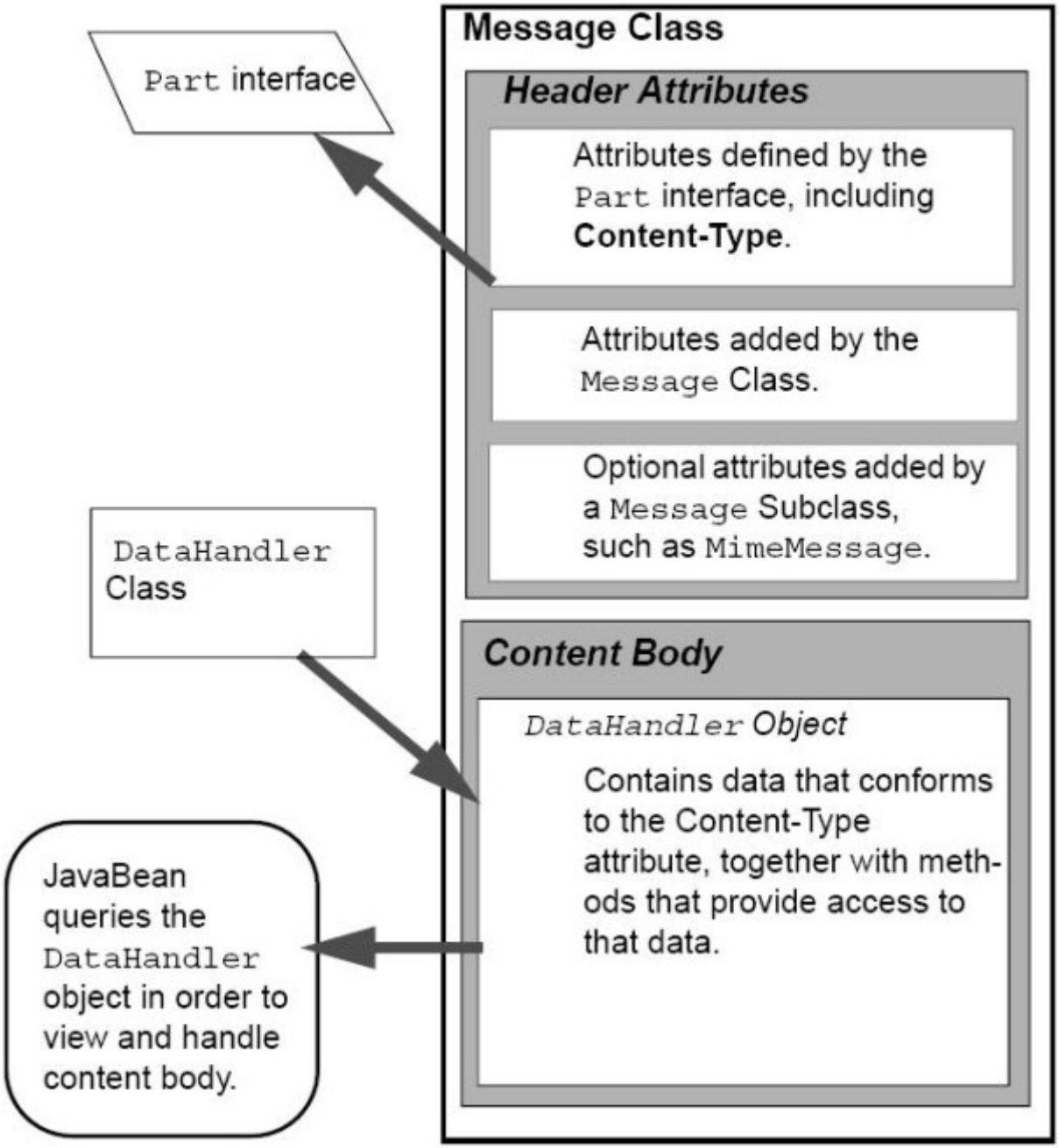
- Adatattribútumokra, és ezek metódusaira

Ezek segítségével férhetünk hozzá az e-mail tartalmához, csatolmányaihoz. Ilyen metódus a `message.getContent()`, melynek visszatérési értéke a fejléctől függően a következő lehet:

`'text/plain'` és `'text/html'` típusú fejléc esetén `String` típusú objektumot kapunk.

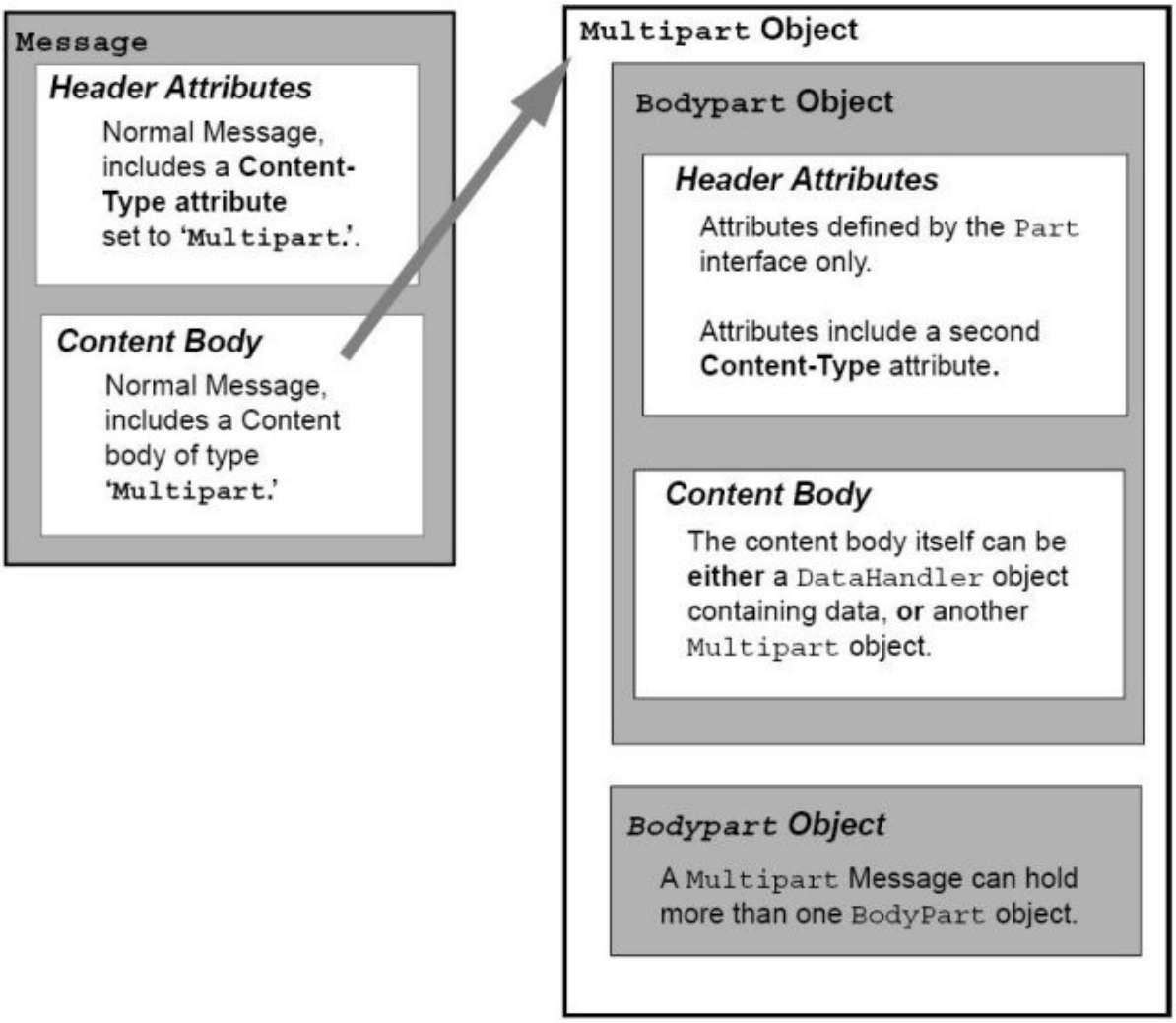
`'application/…'` esetén egy `java.io.InputStream` típusú objektumot kapunk, amelyből közvetlenül kiolvashatjuk a tartalmat.

`'multipart/…'` esetén `javax.mail.internet.MimeMultipart` objektumot kapunk, amelybe beágyazva találhatjuk meg az összetett e-mail egyes `MimeBody` komponenseit.



3. ábra

Volt már szó összetett e-mailekről. Programozói szemszögből tekintve, akkor beszélünk összetett e-mailről, ha az általa reprezentált `Message` objektum `getContent()` metódusa egy `MimeMultipart` típusú objektumot ad vissza. Ez egy konténer osztály, amely meg tudja mondani, hogy hány komponensből (`BodyPart`-ból) tevődik össze az email. Mindegyik komponens rendelkezik saját fejléc attribútumokkal és adattaggal, mivel implementálják a `Part` interfészt. (4. ábra)



4. ábra

2.2 Apache James

Az Apache által fejlesztett James egy nyílt forráskódú, 100%-ban Java alapú SMTP (Simple Mail Transfer Protocol), POP3 (Post Office Protocol version 3) e-mail szerver, illetve NNTP (Network News Transfer Protocol) hírszerver. Alapértelmezés szerint az e-maileket a háttértáron tárolja, de adatbázis támogatása révén lehetőségünk nyílik az e-maileket adatbázisban is tárolni. Én az utóbbi megoldást választottam.

Telepítése nem jelent különösebb problémát, az apache.org oldaláról letöltött állományt kicsomagoljuk, és a `/james-x.x.x/bin/run.bat` állomány segítségével elindítjuk.

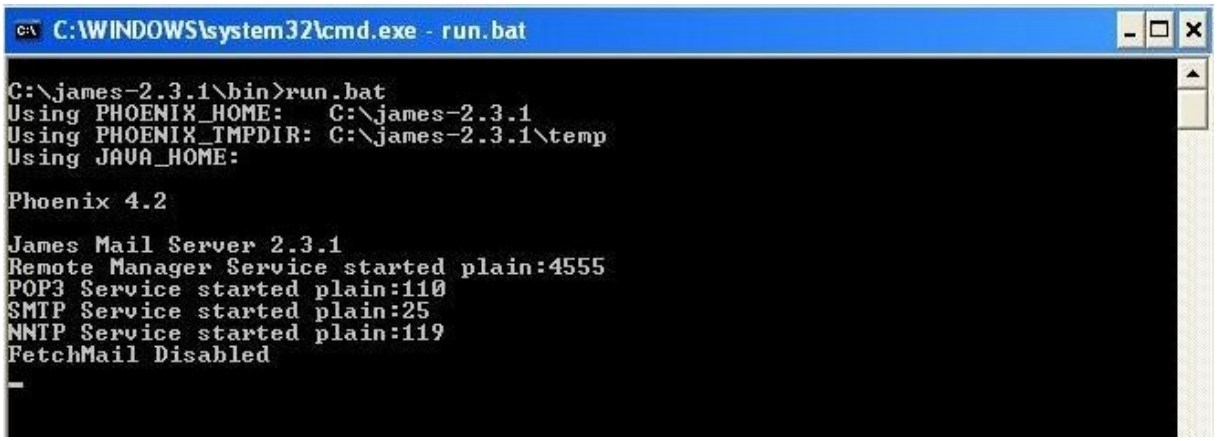
Első indításkor generálódnak le a konfigurációs xml-ek, többek közt a `config.xml`, mely a `/james-x-x-x/apps/james/Sar-inf/` könyvtárban lesz megtalálható. Ebben állíthatjuk be többek közt azt, hogy:

- milyen néven legyen elérhető e-mail szerverünk
  `<servername> szsolt.dyndns.org </servername>`
  (`config.xml` 56.sor)
- adatbázisban szeretnénk tárolni az e-mail-eket (ekkor azonban be kell másolnunk a postgreSQL JDBC drivert a `/james-x-x-x/lib` könyvtárba)

```
<database-connection >
<data-source name="maildb" class="org.apache.james.util.dbcp.JdbcDataSource">
        <driver> org.postgresql.Driver </driver>
        <dburl> jdbc:postgresql://localhost:5432/mail?autoReconnect=true </dburl>
        <user> postgres_mail </user>
        <password> postgres_mail </password>
        <max> 20 </max>
</data-source>
</database-connections>
```

(`config.xml` 1195. sor)

Miután saját elvárásainknak megfelelően konfiguráltuk e-mail szerverünket, elindíthatjuk, melynek eredményeképp a háttérben fog futni, és beérkező e-mailekre fog várni. (5. ábra)



5. ábra

2.3 Apache Tomcat

Az Apache által kifejlesztett Tomcat egy nyílt forráskódú webalkalmazás szerver, amely teljeskörűen implementálja a Sun Microsystems által kidolgozott Java Servlet és

JavaServer Pages specifikációit. Telepítése Windows típusú rendszerekre nagyon egyszerű, telepítő varázsló segít nekünk ebben. Telepítésünket úgy ellenőrizhetjük, hogy a böngészőnkbe beírjuk http://localhost:8080/ címet. Sikeres telepítés esetén az Apache Tomcat üdvözlőlapja jelenik meg. Alapértelmezés szerint Tomcat szerverünk a 8080-as portot fogja figyelni a beérkező HTML kérésekre, ezt megváltoztatni a `/Tomcatx.x/conf/server.xml` konfigurációs xml állományban lehet a következő bejegyzés megváltoztatásával:

```
<Connector port="80" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />
```

(`server.xml` 50.sor)

Tetszés szerint megváltoztathajtjuk webszerverünk nevét (hosztját) is:

```
<Host name="szsolt.dyndns.org"  appBase="webapps"
            unpackWARs="true" autoDeploy="true"
            xmlValidation="false" xmlNamespaceAware="false">
</Host>
```

Miután sikeresen konfiguráltuk webszerverünket, az elkészült webalkalmazásunkat be kell másolnunk a `/Tomcat x.x/webapps/` nevű könyvtárba, és már futtathatjuk is alkalmazásunkat a böngészőnkben.

2.4 PostgreSQL

A PostgreSQL egy nyílt forráskódú, BSD típusú licensz alatt fejlesztett objektum-relációs adatbázis-kezelő rendszer. A MySQL vetélytársaként emlegetik, mivel mindkettőt inkább kisebb alkalmazások, illetve webalkalmazások alá ajánlják. Választásom egyrészt azért esett a PostgreSQL-re, mert mikor még a Haladó Oracle 1 című órán az ORACLE PL/SQL-ben programoztunk, otthon a PostgreSQL PL/pgSQL-jében volt lehetőségem gyakorolni, ugyanis a két nyelv szintaktikája 90%-ban megegyezik. Másrészt a PostgreSQL számos egyéb más kényelmi szolgáltatásokat nyújt (pl. triggerek írása), amit MySQL-ben még jelen pillanatban nem lehet.

Telepítése ugyanolyan egyszerű, mint az Apache Tomcat esetében volt. Miután a *http://postgresql.org* oldaláról letöltöttük az exe kiterjesztésű állományt, telepítő varázsló segít nekünk eligazodni. Ezek után célszerű feltelepíteni még egy kimondottan PostgreSQL-hez fejlesztett *pgAdmin III* nevű programot, amely kényelmes grafikus felületet biztosít az

adatbázissal kapcsolatos műveleteknél. Ugyancsak ingyenes szoftverről van szó, amely letölthető a *http://www.pgadmin.org* című oldalról.

2.5 NetBeans

A NetBeans egy nyílt forráskódú integrált fejlesztői környezet (IDE), amely a Java nyelven alapul. A program grafikus fejlesztőfelületet kínál a különböző alkalmazások, appletek vagy akár JavaBeanek elkészítéséhez, amelynek segítségével könnyebben, gyorsabban tudjuk fejleszteni saját programjainkat. Jelenlegi legfrissebb verzió az 5.5.1-es, amelyet én is használtam a fejlesztés során (a 6.0 még csak béta verzióban érhető el). A NetBeans-hez lehetőségünk van letölteni különböző kiegészítő csomagokat, így pl.:

- NetBeans Mobility Pack – tipikusan JME (Java Micro Edtition) platformra készülő alkalmazások írásában vehetjük hasznát.
- NetBean Enterprise Pack – mely többek közt webalkalmazások, és webszolgáltatások fejlesztését segíti elő.
- NetBeans Visual Web Pack – mely *drag and drop* típusú felületet biztosít web alapú felületfejlesztésre.

3. WebMail alkalmazás tervezése

Napjainkban a különböző tervezőeszközök elterjedésének ellenére is a projektek zöme késik, a leszállított szoftverek nem, vagy pontatlanul felelnek meg a követelményeknek. Ennek az oka egyrészről a szakképzett vezetés hiánya, másrészről pedig az átgondolatlan, elhamarkodott tervezés. Egy rendszerfejlesztési folyamatban a követelményelemzést közvetlenül a tervezésnek kell követnie, így célszerű alaposan átgondolni az elkészítendő szoftver tervezetét, mivel ez az a szakasz, amikor még a legkevesebb erőfeszítéssel javíthatók a hibák.

Nem szabad azonban megfeledkezni arról, hogy a mai szoftverekkel szemben támasztott egyik legfontosabb követelmény a rugalmasság. Ez pedig azt jelenti, hogy nem tervezhetjük meg már az elején a szoftvert A-tól Z-ig, elegendő, ha a kezdetben annak csak a vázlatos keretét adjuk meg. A tervezésnek végig kell kísérnie a teljes fejlesztési folyamatot, ugyanis gyakran arra kényszerülünk, hogy módosítsuk eredeti elképzeléseinket, vagy újabb komponensekkel szeretnénk bővíteni a már meglévő rendszert.

3.1 Követelmények felvázolása

Tervezés előtt tisztáznunk kell, hogy leendő alkalmazásunk milyen alapvető funkciókat lásson el, milyen kényelmi szolgáltatásokat nyújtson a felhasználóknak stb. Mivel projektem elsősorban gyakorlás céljából lett megvalósítva, inkább az alapvető funkciók helyes működésére fektettem nagyobb hangsúlyt. A következő alapfunkciókra gondoltam:

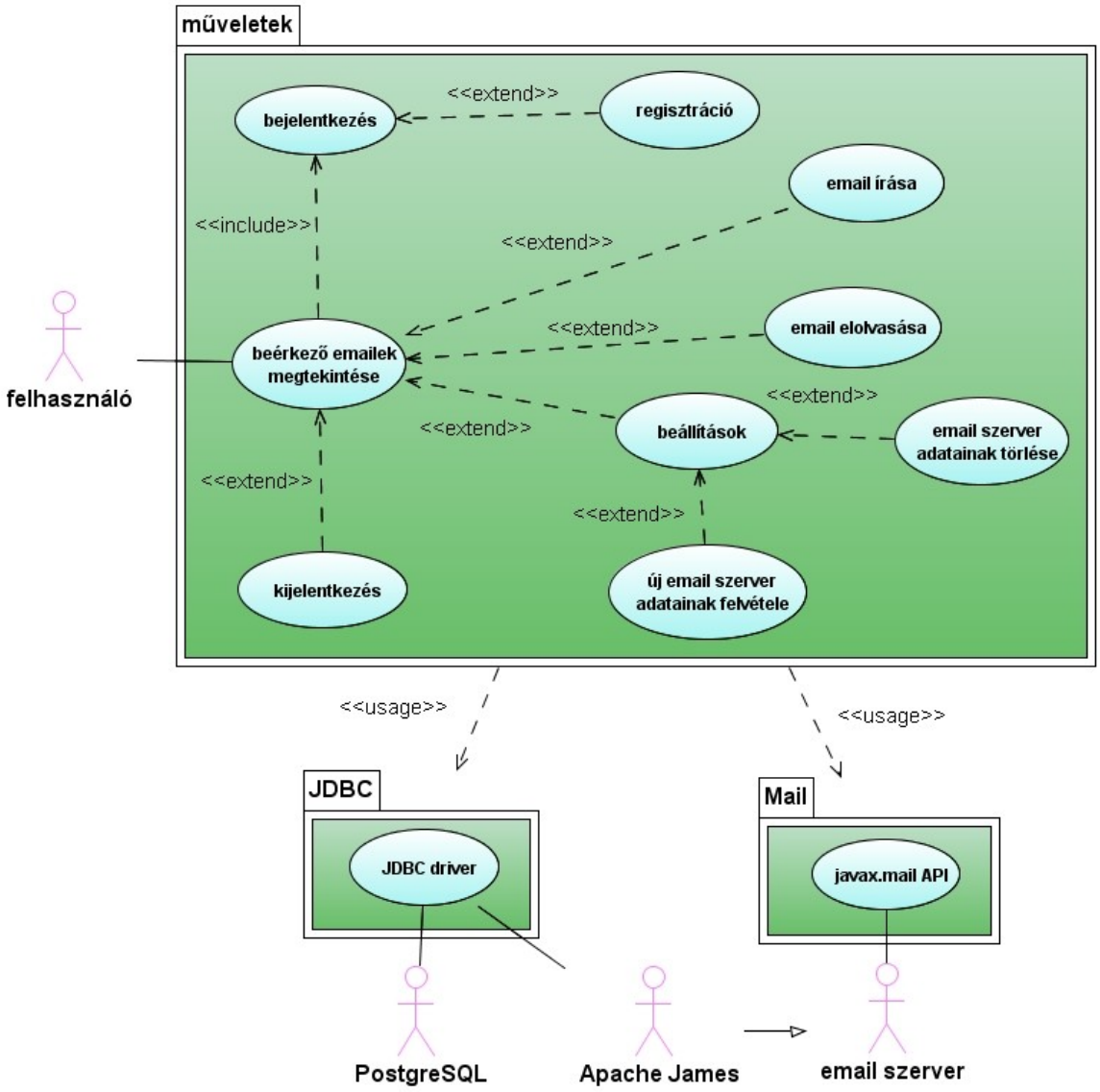
- csak regisztrált felhasználók használhatják az alkalmazást
- e-mailek letöltése különböző szerverekről
- e-mailek törlése
- e-mailek írása és elküldése
- új e-mail szerver adatok felvitelének lehetősége
- biztonságos rendszer megvalósítása, azaz ne lehessen az alkalmazást autentikáció nélkül megkerülni, és illetéktelen adatokhoz hozzáférni

Elsősorban ezekre az alapvető funkciókra fordítottam nagyobb hangsúlyt, és jelen pillanatban kényelmi szolgáltatásokat nem dolgoztam ki, nem integráltam a rendszerbe.

3.2 Használati eset diagram

Miután felvázoltuk a rendszerrel kapcsolatos legfőbb követelményeket, vizuálisan is rögzíthetjük UML (Unified Modeling Language) diagramok segítségével. Azért fontos ezt megtennünk, mert az UML az objektum-orientált programozás szabványos specifikációs nyelve, ezáltal ha csapatmunkában dolgozunk, akkor mások számára is sokkal könnyebb lesz megérteni a rendszer működését, illetve leprogramozását.

Az egyik legfontosabb UML diagram a használati eset diagram (6. ábra) (angol nevén *use case*), amelynek segítségével vizuálisan, ábrák segítségével is definiálhatjuk az alkalmazással szemben támasztott követelményeket.



6. ábra

A fenti használati eset diagram a következő komponensekből tevődik össze:

- *aktorok* (pálcikaemberkék), amelyek általában a felhasználók, de lehetnek egyéb a rendszerhez kapcsolódó külső elemek is. Jelen esetben négy aktort definiáltam: felhasználót, a PostgreSQL adatbázis-kezelő rendszert, egy általános e-mail szerver szerepet, és ennek egy pontosítását, az Apache James e-mail szervert.

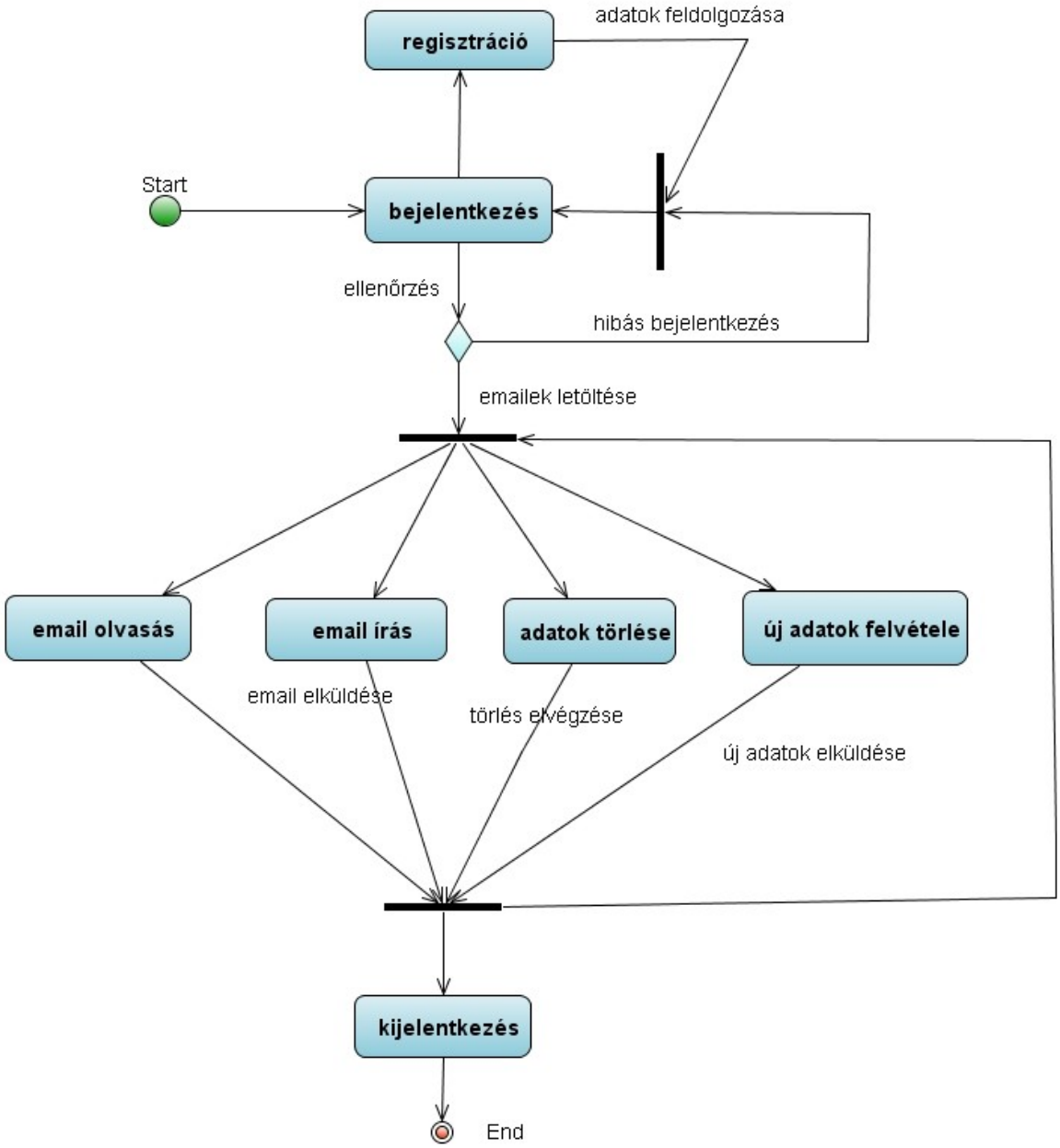
- *használati esetek* (világoskék hátterű ellipszisek), melyek a rendszer funkcióit, külső kapcsolódási pontjait írják le. Vannak kiemelt szerepkörű használati

esetek, ilyen például a *beérkező e-mailek megtekintése*, *JDBC driver*, és a *javax.Mail Api* használati esetek, melyekhez aktorok kapcsolódnak.

- <<*extends*>> sztereotípiák (szaggatott vonalú nyilak), melyeket két használati eset között definiálunk. Ilyen például a *beérkező e-mailek megtekintése* és az *e-mail írása* használati esetek közti sztereotípia, amely azt jelenti, hogy az *e-mail írása* használati eset által képviselt viselkedés opcionális a *beérkező e-mailek megtekintése* használati esetre nézve.
- <<*include*>> sztereotípiák (szaggatott vonalú nyilak), melyeket két használati eset között definiálunk. Ilyen pl. a *beérkező e-mailek megtekintése* és a *bejelentkezés* használati esetek közti sztereotípia, mely azt jelenti, hogy a *bejelentkezés* használati eset által képviselt viselkedést magába foglalja a *beérkező e-mailek megtekintése* használati eset.
- <<*usage*>> sztereotípiák (szaggatott vonalú nyilak), jelen esetben arra utalnak, hogy a rendszer felhasznál már előre megírt komponenseket, mint pl. *javax.mail API* és a *JDBC*.

3.3 Állapot-átmenet diagram

Az állapot-átmenet diagram (7. ábra) segítségével nyomon követhetjük, hogy milyen folyamatok fognak a rendszeren belül lezajlani. A rendszert futtatása során állapotokkal (lekerekített téglalapok) jellemezhetjük, és egyik állapotból a másikba egy bizonyos művelet, esemény (folytonos vonalú nyíl) hatására megy át. Szerkezete nagyon hasonlít a folyamatábrához.



7. ábra

4. WebMail alkalmazás bemutatása

4.1 WebMail felhasználói szemszögből

Az alkalmazás elindítása után rögtön egy bejelentkezési oldallal találkozunk (8. ábra). Nem léphetünk be a rendszerbe egészen addig, míg nem azonosítjuk magunkat egy felhasználói név-jelszó párossal. Amennyiben új felhasználóról van szó, regisztrálhat a

registration linkre kattintva (9. ábra), kitölti értelemszerűen a mezőket, majd elküldi feldolgozásra a szervernek. Hibaüzenet fog minket figyelmeztetni, ha valamelyik mezőt véletlenül üresen hagytuk, illetve ha már létező felhasználói nevet szeretnénk újból regisztrálni. Sikeres regisztráció után a rendszer visszairányít a bejelentkező oldalra, és várja bejelentkezésünket. Miután sikeresen be is jelentkeztünk (10. ábra), megismerkedhetünk az alkalmazás lehetőségeivel.

Ahhoz, hogy e-maileket tudjunk letölteni, fel kell vennünk új e-mail szerver adatokat. Ezt az oldal bal alsó sarkában tehetjük meg az *add new server* linkre kattintva (11. ábra). Fontos, hogy minden mezőt helyesen töltsünk ki, ugyanis ezekkel az adatokkal fog majd az alkalmazás csatlakozást kísérelni az e-mail szerverhez. Ha elgépeltünk valamit a felhasználói név-jelszó párosban, a *Read* gomb megnyomása után *Authentication failed Bad username or password* hibaüzenetet fogunk kapni (12. ábra). Ha a hiba más jellegű (például nem elérhető a szerver, vagy elgépeltük az *smtp host*-ot), akkor *Unable to connect. Please try again* hibaüzenet fog megjelenni ugyanott. Ha a sorozatos próbálkozások ellenére sem sikerül kapcsolatot létesíteni az e-mail szerverrel, akkor nagy valószínűséggel mi gépeltünk el valamit, miközben az új szerver adatait vittük fel. Ilyen esetben az oldal bal alsó sarában lévő *delete server* linkre kell kattintanunk (13. ábra), majd miután kiválasztottuk a servert, törölhetjük.

Miután újból felvettük a servert, de már helyesen, a *Read* gomb megnyomása után az alkalmazás elkezdi letölteni és egyből kilistázni az e-maileket (14. ábra). Itt szeretném megjegyezni, hogy POP3 protokollal dolgoztam, ami egyben azt jelenti, hogy a letöltött e-mailek másolatok, az eredetik megmaradnak a szerveren. Egy e-mail elolvasásához egyszerűen csak kattintsunk valamelyik e-mailre. Lehetőségünk van csatolmányokat is letölteni, amennyiben érkezett az e-mailben ilyen. Ezt az oldal legalján tehetjük meg az *Attachments* után található linkekre kattintva. E-mailek törlése esetén jelöljük ki, hogy melyeket szeretnénk törölni, majd az oldal bal oldalán kattintsunk a *Delete selected emails* linkre (15. ábra). Egy felugró ablak fog megerősítést kérni tőlünk a művelet elvégzésére, és amennyiben ezt jóváhagytuk, az alkalmazás véglegesen törölni fogja a kijelölt e-maileket a szerverről.

E-mailek írására is van lehetőségünk, amennyiben az oldal lévő *Write email* linkre kattintunk (16. ábra). Egy legördülő listából ki kell választanunk, hogy mely e-mail szerver segítségét szeretnénk segítségül kérni e-mailünk elküldésére. Amint elkészültünk e-

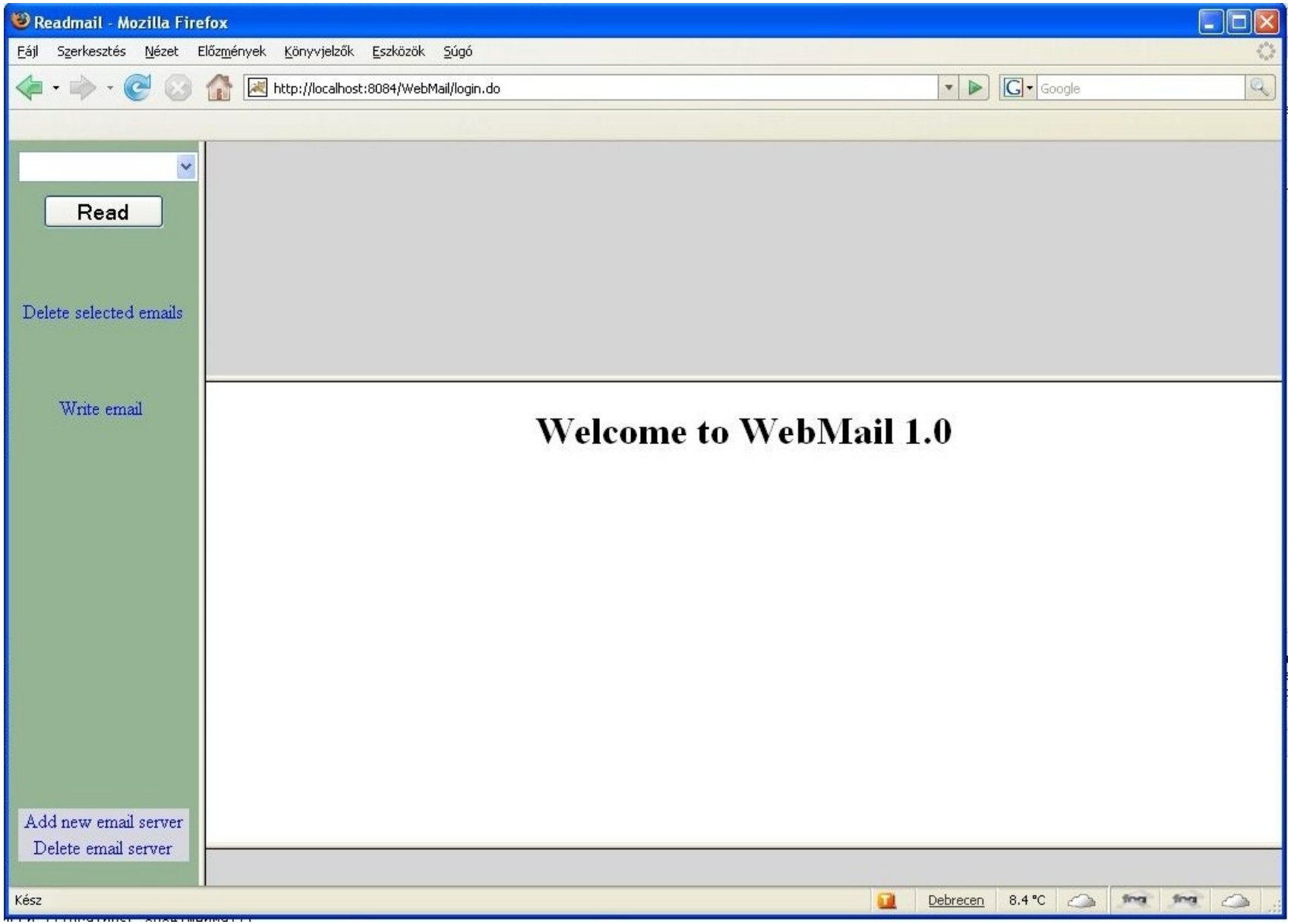
mailünkkel, a Send gombra kattintva elküldhetjük a címzettnek. Fejlesztés jelen állapota szerint csak *text/plain* szövegtípusú e-mailek elküldése lehetséges.

Login - Mozilla Firefox
Fájl Szerkesztés Nézet Előzmények Könyvjelzők Eszközök Súgó
http://localhost:8084/webMail/
Google
Username: zsolt
Password:
registration
Login
Kész
Debrecen
8.3 °C

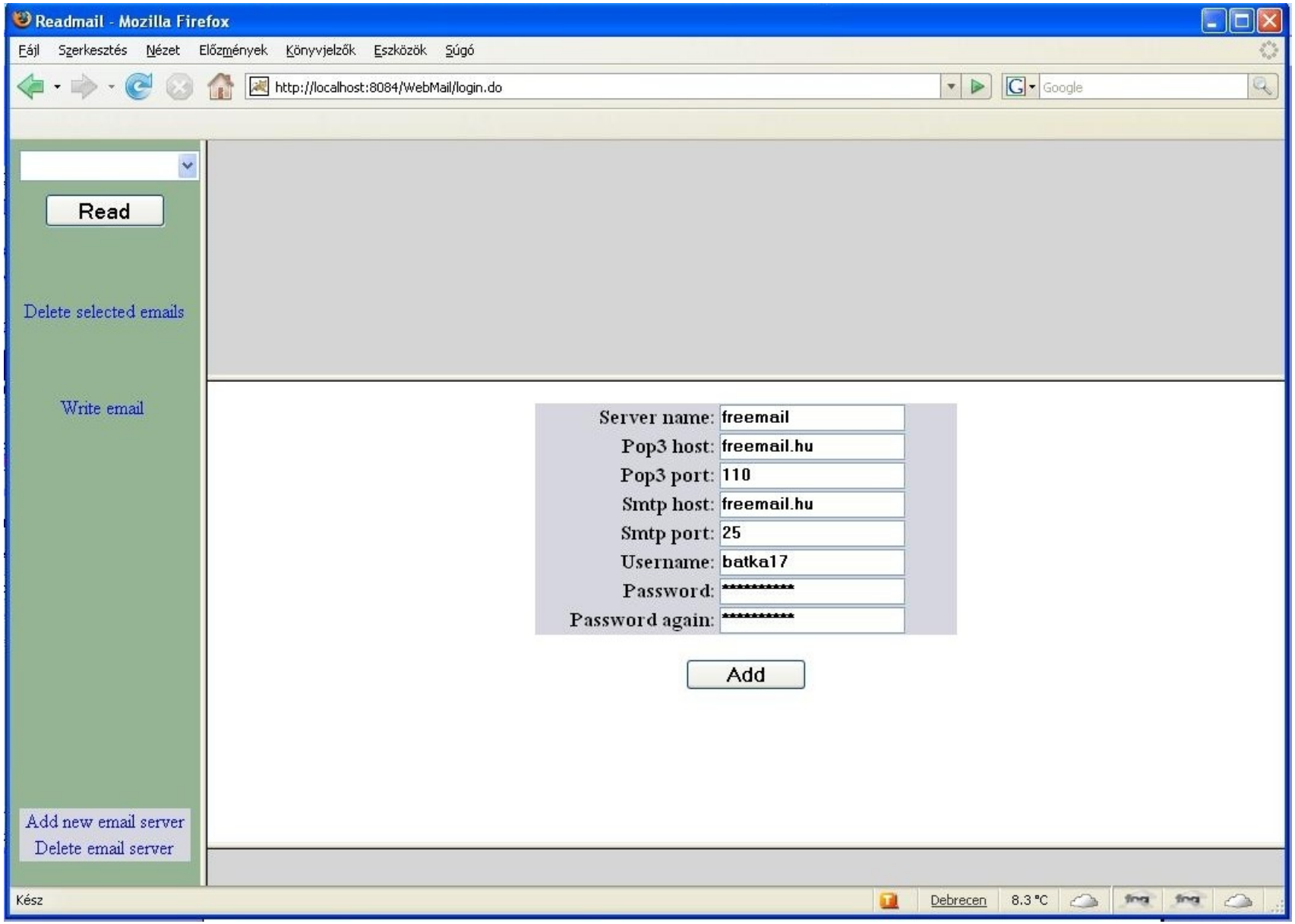
8. ábra

Registration - Mozilla Firefox
Fájl Szerkesztés Nézet Előzmények Könyvjelzők Eszközök Súgó
http://localhost:8084/webMail/registration.jsp
Google
Username: zsolt
Password:
Password again:
Submit
Kész
Debrecen
8.3 °C

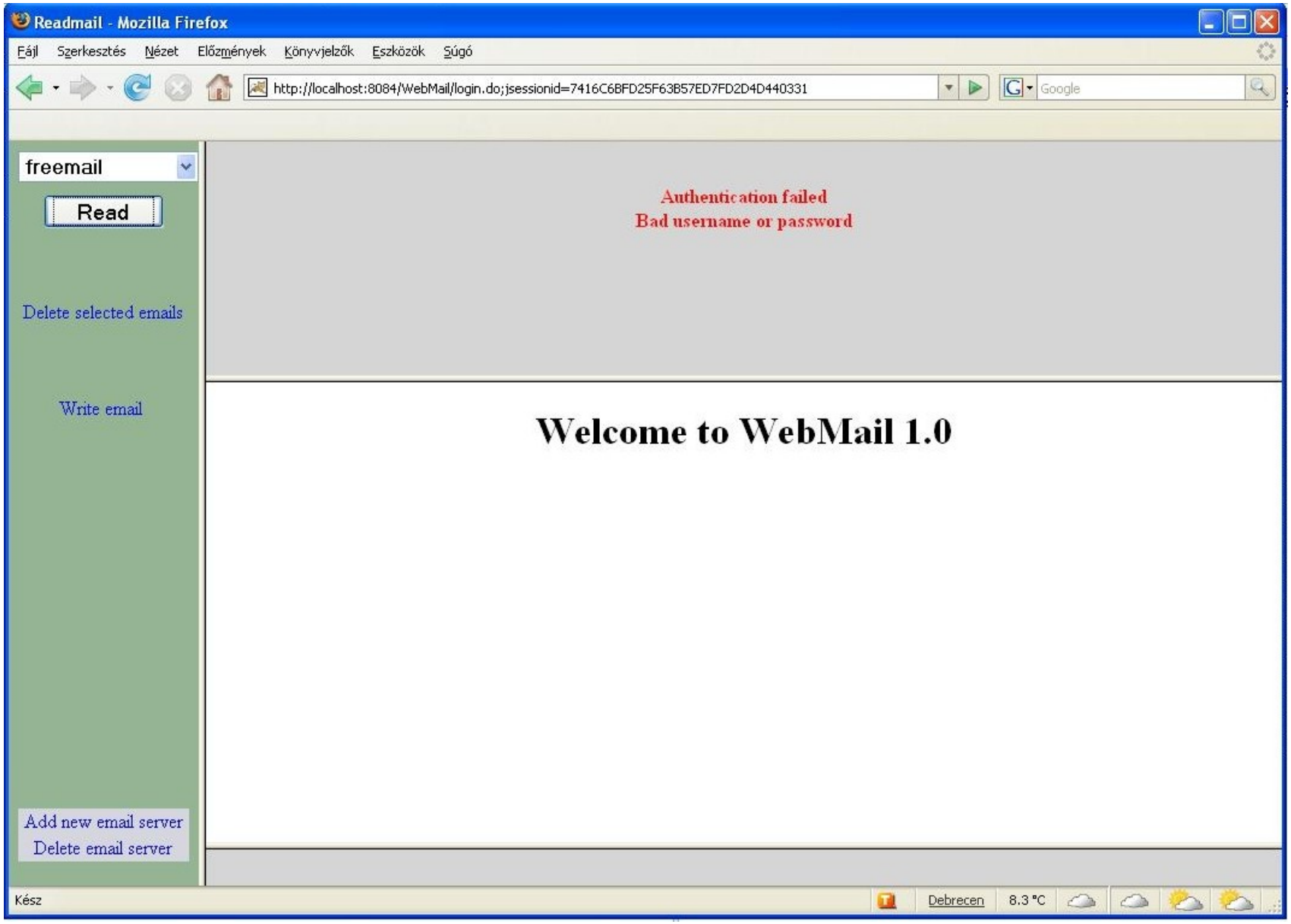
9. ábra


Readmail - Mozilla Firefox
Fájl Szerkesztés Nézet Előzmények Könyvjelzők Eszközök Súgó
http://localhost:8084/WebMail/login.do
Google
Read
Delete selected emails
Write email
Add new email server
Delete email server
Welcome to WebMail 1.0
Kész
Debrecen
8.4 °C

10. ábra


Readmail - Mozilla Firefox
Fájl Szerkesztés Nézet Előzmények Könyvjelzők Eszközök Súgó
http://localhost:8084/WebMail/login.do
Google
Read
Delete selected emails
Write email
Server name: freemail
Pop3 host: freemail.hu
Pop3 port: 110
Smtp host: freemail.hu
Smtp port: 25
Username: batka17
Password: **********
Password again: **********
Add
Add new email server
Delete email server
Kész
Debrecen
8.3 °C

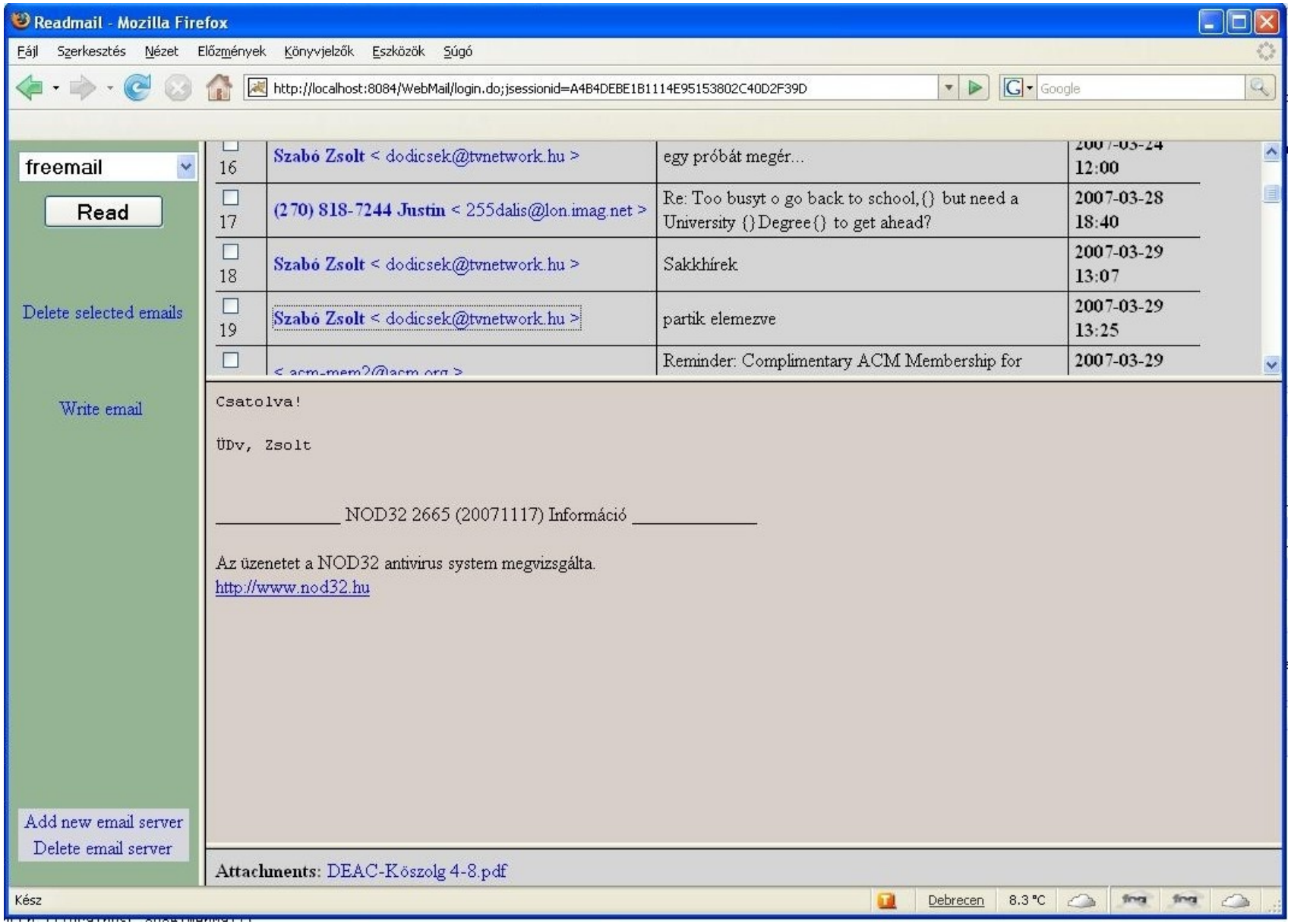
11. ábra


Readmail - Mozilla Firefox
Fájl Szerkesztés Nézet Előzmények Könyvjelzők Eszközök Súgó
http://localhost:8084/WebMail/login.do;jsessionid=7416C6BFD25F63B57ED7FD2D4D440331
Google
freemail
Read
Delete selected emails
Write email
Add new email server
Delete email server
Authentication failed
Bad username or password
Welcome to WebMail 1.0
Kész
Debrecen
8.3 °C

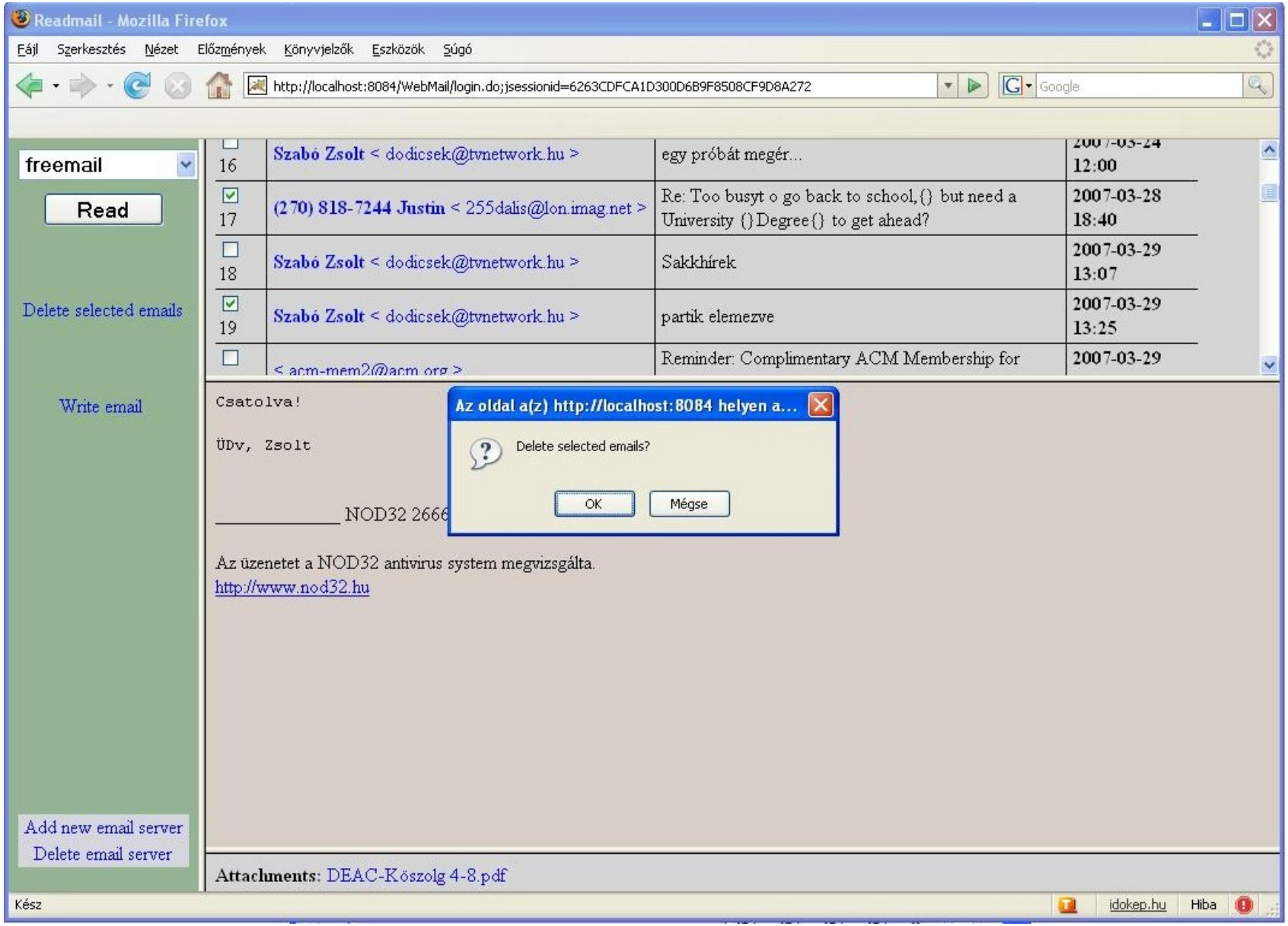
12. ábra

Readmail - Mozilla Firefox
Fájl Szerkesztés Nézet Előzmények Könyvjelzők Eszközök Súgó
http://localhost:8084/webMail/login.do;jsessionid=7416C6BFD25F63B57ED7FD2D4D440331
Google
freemail
Read
Delete selected emails
Write email
Add new email server
Delete email server
freemail
Delete
Kész
Debrecen
8,3 °C

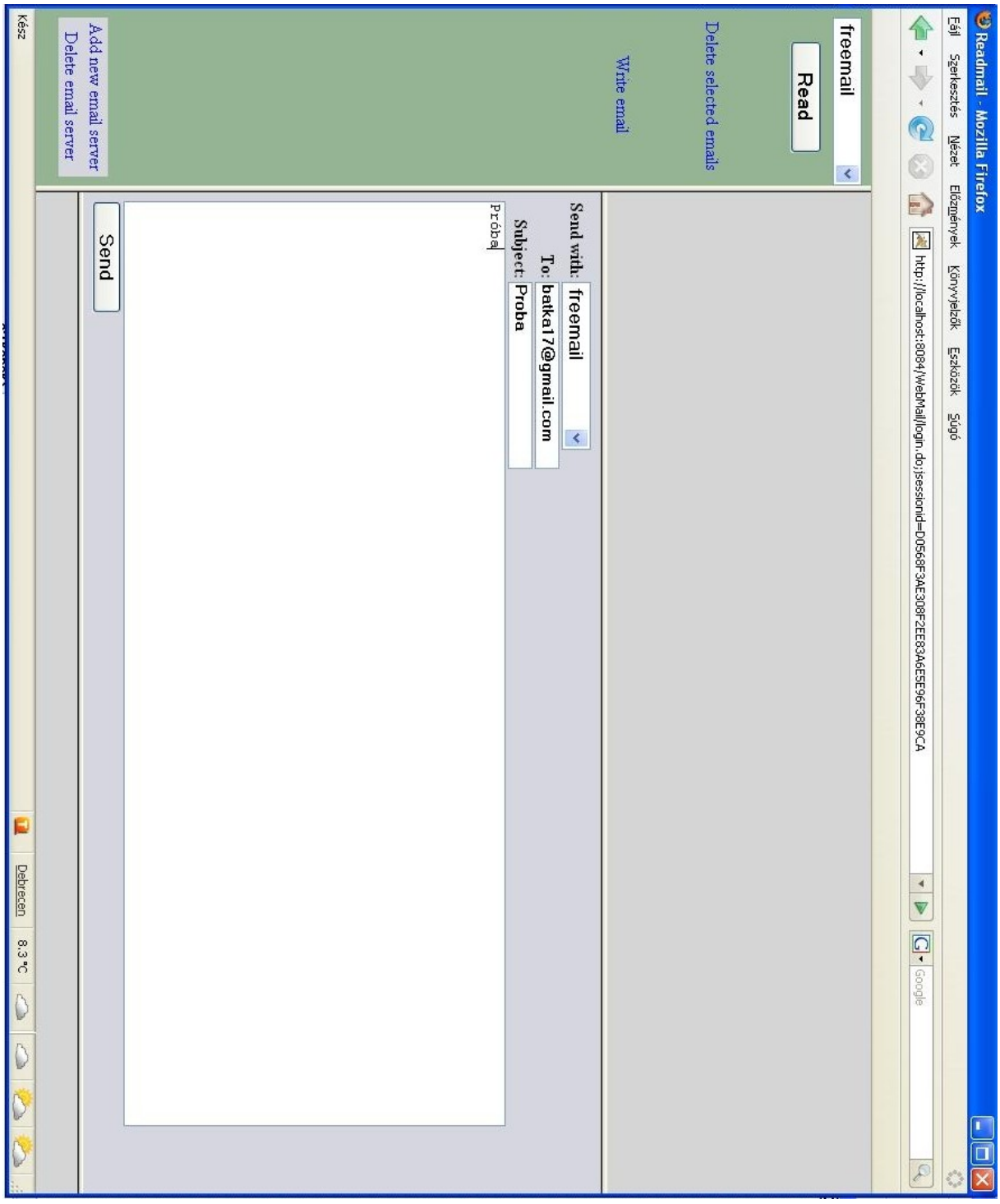
13. ábra


Readmail - Mozilla Firefox
Fájl Szerkesztés Nézet Előzmények Könyvjelzők Eszközök Súgó
http://localhost:8084/WebMail/login.do;jsessionid=A4B4DEBE1B1114E95153802C40D2F39D
freemail
Read
Delete selected emails
Write email
Add new email server
Delete email server
16 Szabó Zsolt < dodicsek@tvnetwork.hu > egy próbát megér... 12:00
17 (270) 818-7244 Justin < 255dalis@lon.imag.net > Re: Too busyt o go back to school,{} but need a University {}Degree{} to get ahead? 2007-03-28 18:40
18 Szabó Zsolt < dodicsek@tvnetwork.hu > Sakkhírek 2007-03-29 13:07
19 Szabó Zsolt < dodicsek@tvnetwork.hu > partik elemezve 2007-03-29 13:25
Reminder: Complimentary ACM Membership for 2007-03-29
Csatolva!
ÜDv, Zsolt
NOD32 2665 (20071117) Információ
Az üzenetet a NOD32 antivirus system megvizsgálta.
http://www.nod32.hu
Attachments: DEAC-Köszolg 4-8.pdf
Kész
Debrecen
8.3 °C

14. ábra


Readmail - Mozilla Firefox
Fájl Szerkesztés Nézet Előzmények Könyvjelzők Eszközök Súgó
http://localhost:8084/WebMail/login.do;jsessionid=6263CDFCA1D300D6B9F8508CF9D8A272
freemail
Read
Delete selected emails
Write email
Add new email server
Delete email server
16 Szabó Zsolt < dodicsek@tvnetwork.hu > egy próbát megér... 12:00
17 (270) 818-7244 Justin < 255dalis@lon.imag.net > Re: Too busyt o go back to school,{} but need a University {}Degree{} to get ahead? 2007-03-28 18:40
18 Szabó Zsolt < dodicsek@tvnetwork.hu > Sakkhírek 2007-03-29 13:07
19 Szabó Zsolt < dodicsek@tvnetwork.hu > partik elemezve 2007-03-29 13:25
< acm-mem2@acm.org > Reminder: Complimentary ACM Membership for 2007-03-29
Csatolva!
ÜDv, Zsolt
NOD32 2666
Az oldal a(z) http://localhost:8084 helyen a...
Delete selected emails?
OK
Mégse
Az üzenetet a NOD32 antivirus system megvizsgálta.
http://www.nod32.hu
Attachments: DEAC-Köszolg 4-8.pdf
Kész
idokep.hu
Hiba

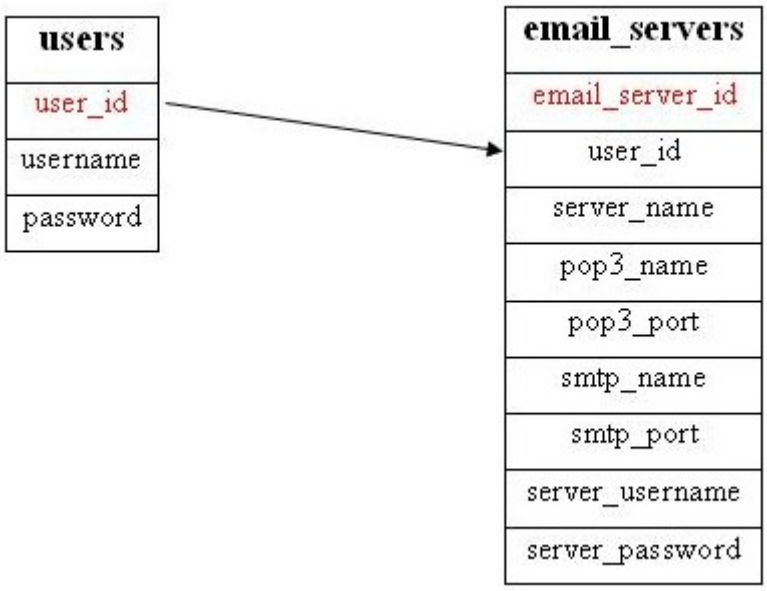
15. ábra


Readmail - Mozilla Firefox
Fájl Szerkesztés Nézet Előzmények Könyvjelzők Eszközök Súgó
http://localhost:8084/WebMail/login.do;jsessionid=D0568F3AE308F2EE83A6E5E96F38E9CA
freemail
Read
Delete selected emails
Write email
Add new email server
Delete email server
Send with: freemail
To: batka17@gmail.com
Subject: Proba
Próba
Send
Kész
Debrecen
8.3 °C

16. ábra

4.2 Webmail programozói szemszögből

4.2.1 Adatbázis struktúra



17. ábra

Adatbázis struktúrám egyszerű, egyéb adatok letárolását nem tartottam szükségesnek.
Az adatbázis részeként létrehoztam két tárolt függvényt:

- `login_check(_username character varying, _password character varying)`, mely a bejelentkezésnél a felhasználó által megadott felhasználói név és jelszó páros létezését ellenőrzi, és visszaadja, hogy hány darab ilyen előfordulást talált. Nyilván, ha 0 darabot talált, akkor illetéktelen felhasználóról van szó, 1-nél nem adhat vissza soha.

```
CREATE OR REPLACE FUNCTION
login_check(_username character varying, _password character varying
RETURNS integer AS
$BODY$declare
    user_count integer default 0;
begin
    select count(*) into user_count from users
    where username=_username and password=_password;
    return user_count;
end;$BODY$
LANGUAGE 'plpgsql' VOLATILE;
```

- `registration_check(_username character varying)`, mely a regisztráció során megadott felhasználói név létezését ellenőrzi. Visszatérési értéke 0 vagy 1, attól függően, hogy van-e már ilyen felhasználói név az adatbázisban. Ha létezik, akkor egy hibaüzenet révén tudatjuk a felhasználóval, hogy válasszon más felhasználói nevet.

```
CREATE OR REPLACE FUNCTION registration_check(_username character
varying)
RETURNS integer AS
$BODY$declare
    user_count integer default 0;
begin
    select count(*) into user_count from users where
username=_username;
    return user_count;
end;$BODY$
LANGUAGE 'plpgsql' VOLATILE;
```

4.2.2 Java osztályok ismertetése

Java osztályokat szerepük betöltése alapján négy csomagba szerveztem. Ezek a következők:

- `db`, adatbázis műveletekkel kapcsolatos osztályokat tartalmaz
- `mail`, általános e-mail-kezelő osztályok
- `struts.bean`, `StrutsActionBean` típusú osztályok gyűjtőhelye
- `struts.action`, `StrutsAction` típusú osztályok gyűjtőhelye

db csomag

A db csomagban egyetlen Java osztály található, a `DatabaseOperations.java`. Ebben statikus metódusok formájában kimondottan a WebMail számára leprogramozott adatbázis műveletek találhatók. Ezeknek a metódusoknak a specifikációi a következők:

`public class DatabaseOperations`

`private static Connection getConnection() throws Exception {…}`

adatbázis kapcsolódáshoz szükséges információkat tároló objektumot ad vissza. Csak osztályon belül hívható.

```
public static boolean loginCheck(String username, String password) throws
Exception {…}
```

a `ogin_check` nevű tárolt eljárást hívja meg a fenti paraméterekkel. Visszatérési értéke `true`, ha a tárolt eljárás 1-et ad vissza, különben `false`.

```
public static boolean registrationCheck(String username) throws Exception
{…}
```

a `registration_check` nevű tárolt eljárást hívja meg a fenti paraméterrel. Visszatérési értéke `true`, ha a tárolt eljárás 0-át ad vissza.

```
public static int selectUserID(String username) throw Exception {…}
```

felhasználói név alapján kikeresi a hozzátartozó azonosítót.

```
public static HashMap selectServers(int userId) throws Exception {…}
```

felhasználói azonosító alapján kikeresi az összes hozzátartozó e-mail szerver adatot. Ezt egy `HashMap` típusú objektumban adja vissza, melynek kulcs (key) része a szerver neve, értéke (value) pedig egy `mail.Server` típusú objektum (leírását lásd lentebb).

```
public static void insertUser(String username, String password) throws
Exception {…}
```

a fenti 2 paraméterrel beszúr egy új sort a `users` adattáblába.

```
public static void insertServer(int userId, Server newServer) throws
Exception {…}
```

a fenti paraméterekkel beszúr egy új sort az email_servers adattáblába.

```
public static void deleteServer(String emailServerId) throws Exception {…}
```

paraméterként megadott azonosító alapján törli az email szerver adatait az email_servers adattáblából.

mail csomag

A `mail` csomagban 3 Java osztály található, ezek név szerint: `Attachment.java`, `ExtractMessage.java`, `Server.java`. Ezek az osztályok általános, egy e-mail kezeléséhez szükséges műveleteket definiálnak.

`public class Attachment`

ez a JavaBean osztály egy e-mailhez kapcsolódó csatolmányokról tárol információt, illetve magát a csatolmányt is tárolja.

`private String fileName;`

a csatolt állomány nevét tárolja

`private Enumeration headers;`

ez az attribútum tárolja a csatolmány fejléceit. Ilyen pl. `'Content-Type'` fejléc, mely a csatolmány típusára utal, vagy `'Content-Transfer-Encoding'` fejléc, mely a csatolmány kódolási eljárását nevezi meg.

`private InputStream data;`

egy olyan I/O csatornát képvisel, melyről magát a csatolt állományt olvashatjuk ki.

Az osztály további részét képezik a fenti attribútumokat lekérdező- (getter), illetve beállító (setter) metódusok is. Ezeknek ismertetését nem tartottam szükségesnek.

`public class ExtractMessage`

ez az osztály végzi el a lényeges információk kinyerését egy javax.mail.Message típusú üzenetből, gondolok itt magára az üzenetre, és a hozzátartozó csatolmány(ok)ra.

`private Message message;`

ezt az objektumot fogja megkapni az osztály példányosításakor, ebből fogom kinyerni magát az üzenetet, és esetleges csatolmányait.

`private String messageText;`

ez fogja a kinyert üzenetet tárolni.

`private Vector attachments;`

ebben a Vector adatszerkezetben Attachment típusú objektumokat fogok tárolni.

```
private void extractMessage() {…}
private void extractMessage(MimeMultipart mp) throws Exception {…}
```

az első metódust már a konstruktorban meghívom, az attribútumok inicializálása után. A `message` objektumot tartalmilag fogja megvizsgálni. Ha egyszerű `message` objektumról van szó, azaz a `'Content-Type'` fejlécében `'text/plain'` vagy `'text/html'` szerepel, akkor csak szöveges üzenet része van az e-mailnek. Összetett üzenetről akkor beszélünk, ha a `'Content-Type'` fejlécében szerepel a `'multipart'` szó. Ekkor hívódik meg a 2. metódus, mely ezt az összetett üzenetet feldarabolja. Mivel elméletileg ez az összetettség tetszőleges mélységű lehet, ezért ha a 2. metódus

újabb összetett részt talál, akkor saját magát fogja meghívni ezzel az újonnan talált összetett résszel paraméterezve.

```
private String transform(String s) {...}
```

adott esetben egy email üzenetből kétfajta ugyanazt az információt tartalmazó szöveges üzenetet nyerhetünk ki. Az egyik a `'text/plain'` típusú, mely csupán szöveget tartalmaz, a másik a `'text/html'` típusú, mely a HTML-ben jól ismert tageket is tartalmazhat. A két különböző típusú szöveget másképp kell feldolgozni, illetve megjeleníteni, ezért én úgy döntöttem, hogy ha egy `'text/plan'` típusú szöveggel találkozok, akkor minimális változtatással `'text/html'` típusú szöveggé alakítom át azáltal, hogy az újsor karaktereket lecserélem `<br/>` karakterláncokra. Ez a metódus ezt a műveletet hajtja végre.

`public class Server`

---

ez egy olyan JavaBean, mely az adatbázisból kinyert szerverinformációkat tárolja le. Közvetlenül a `DatabaseOperation` osztály `selectServers` statikus metódusa használja fel.

```
private String serverName;
```

email szerver neve (pl. freemail.hu)

```
private String pop3Name;
```

a szerver pop3 kiszolgálójának neve. (pl. freemail esetén freemail.hu)

```
private String pop3Port;
```

a szerver pop3 kiszolgálójának portja. Erre a porta csatlakozva fogom tudni letölteni az e-maileket. Általában 110 szokott lenni, azonban ez szolgáltató-függő.

```
private String smtpName;
```

a szerver smtp kiszolgálójának neve. (pl. freemail esetén freemail.hu)

```
private String smtpPort;
```

a szerver smtp kiszolgálójának portja. Ezen a porton tud egy email szerver emailt fogadni, illetve más email szerver felé e-mailt küldeni. Általában ez 25 szokott lenni.

```
private String serverUsername;
private String serverPassword;
```

a szerverhez való kapcsolódáskor szükséges felhasználói név-jelszó páros.

```
public class UsernameInUseException extends Exception
```

Ez egy olyan egyszerű kivétel, mely a `RegistrationAction execute` metódusának futása során keletkezhet, annak függvényében, hogy a regisztrálni kívánt felhasználónév létezik-e már az adatbázisban.

```
private String username;
```

egy olyan felhasználói nevet tárolunk ebben, ami már az adatbázisban létezik.

struts.bean csomag

Ebben a csomagban azok a JaveBeanek találhatók meg, melyek egy formból érkező adatokat hivatottak eltárolni.

`public class LoginActionForm extends org.apache.struts.action.ActionForm`

```
private String username;
private String password;
```

ez a Bean a bejelentkező oldalon megadott felhasználói név-jelszó párost tárolja el.

```
public class RegistrationActionForm extends
org.apache.struts.action.ActionForm
private String username;
private String password;
```

ez a Bean a regisztrációs oldalon megadott felhasználói név-jelszó párost tárolja el.

`public class SelectActionForm extends org.apache.struts.action.ActionForm`

```
private String selected;
```

ennek a Beannek az a feladata, hogy egy legördülő listából kiválasztott elemet eltárolja. Ilyen legördülő listával találkozhatunk sikeres bejelentkezésünk után, az üdvözlőlap bal felső sarkában.

`pulic class NewServerActionForm extends org.apache.struts.action.ActionForm`

```
private String serverName;
private String pop3Host;
private String pop3Port;
private String smtpHost;
private String smtpPort;
private String username;
private String password;
```

ez a Bean fogja ideiglenesen eltárolni az újonnan megadott szerveradatokat.

```
pulic class DeleteEmailActionForm extends org.apache.struts.action.ActionForm
```

```
private String[] deleted;
```

miután sikeresen ki tudtuk listázni egy bizonyos szerverről az e-mailjeinket, minden egyes listaelem bal oldalán észrevehető egy checkbox, melyet ha bejelölünk, és a Delete selected e-mails linkre kattintunk, akkor az alkalmazás végleg törölni fogja ezt az e-mailt. Tehát a checkboxnak jelző szerepe van. Ezzel jelezzük, hogy mely e-maileket szeretnénk törölni. A deleted attribútum egy olyan String tömb, melynek elemei sorszámok, a kilistázott és törlésre bejelölt e-mailek sorszámát tartalmazza. Jelen pillanatban ennek a lekérdezése csak String tömb formájában érhető el, természetesen sokkal kézenfekvőbb lenne egy primitív típusú int tömb kezelése.

```
pulic class SendEmailActionForm extends org.apache.struts.action.ActionForm
```

```
private String selected;
```

egy legördülő listából kiválasztott e-mail szerver nevét fogja letárolni, melynek segítségével szeretnénk elküldeni az újonnan szerkesztett e-mailünket.

```
private String to;
```

a címzett e-mail címét tárolja le.

```
private String subject;
```

az e-mail tárgyát tárolja le.

```
private String text;
```

az e-mail üzenetét tárolja le.

struts.action csomag

Ebben a csomagban találhatók azok a Java osztályok, melyek `struts.bean.*` típusú objektumokat dolgozzák fel. Nézzük ezeket sorban:

```
public class LoginAction extends Action

public ActionForward execute(ActionMapping mapping, ActionForm  form,
HttpServletRequest request, HttpServletResponse response) throws Exception
{
try {
      LoginActionForm loginForm = (LoginActionForm)form;
       if (DatabaseOperations.loginCheck(loginForm.getUsername(),
```

```
      loginForm.getPassword())== false)  throw new
AuthenticationFailedException();
      int userId =
DatabaseOperations.selectUserId(loginForm.getUsername());
      HashMap servers = DatabaseOperations.selectServers(userId);
      request.getSession().setAttribute("userLogin", "true");
      request.getSession().setAttribute("userId", new Integer(userId));
      request.getSession().setAttribute("servers", servers);
}
catch (AuthenticationFailedException afe) {
      ActionMessages a = new ActionMessages();
      a.add("login.error", new ActionMessage("login.error"));
      saveErrors(request, a);
      return mapping.findForward(FAILURE);
}
catch (Exception e) {
      e.printStackTrace();
      ActionMessages a = new ActionMessages();
      a.add("unable.to.connect", new ActionMessage("unable.to.connect"));
      saveErrors(request, a);
      return mapping.findForward(FAILURE);
}
return mapping.findForward(SUCCESS);
}
```

Ennek az osztálynak a segítségével végzem el a bejelentkeztetést. Itt hívom segítségül a `DatabaseOperation` osztály `loginCheck` metódusát, melynek visszatérési értékétől függően lesz sikeres, vagy sikertelen a bejelentkezés. Sikeres bejelentkezést követően az adatbázisból lekérdezem, hogy a megadott username-hez milyen userId tartozik, majd ez alapján ugyancsak az adatbázisból leválogatom, hogy mely e-mail szerver adatok tartoznak ehhez a felhasználóhoz. Az így kinyert adatokat a session attribútumban eltárolom, mint session változók, majd tovább léptetem az oldalt az üdvözlőoldalra. Ha sikertelen a bejelentkezés, akkor kivételt dobok, amit a kivételkezelőben lekezelek. Ilyen esetekben a kivételkezelő a bejelentkező oldalra irányít vissza.

```
public class RegistrationAction extends Action

public ActionForward execute(ActionMapping mapping, ActionForm  form,
```

```
HttpServletRequest request, HttpServletResponse response) throws Exception
{
try {
      RegistrationActionForm raf = (RegistrationActionForm) form;
      if (DatabaseOperations.registrationCheck(raf.getUsername()) == true)
      throw new UsernameInUseException(raf.getUsername());
      DatabaseOperations.insertUser(raf.getUsername(), raf.getPassword());
}
catch (UsernameInUseException uiue) {
      ActionMessages a = new ActionMessages();
      a.add("username.in.use", new ActionMessage("username.in.use",
            uiue.getUsername()));
      saveErrors(request, a);
      return mapping.findForward(FAILURE);
}
return mapping.findForward(SUCCESS);
}
```

Ez az osztály kezeli egy új felhasználó regisztrálását. Mielőtt még felviszem az adatokat az adatbázisba, leellenőrzöm, hogy szerepel-e ilyen felhasználói név már az adatbázisban. Ha szerepel akkor kivételt dobok, és az oldalon figyelmeztetem a felhasználót, hogy válasszon más nevet. Ha nem szerepel, egyszerűen felveszek egy új sort az adatbázis megfelelő táblájába.

```
public class NewServerAction extends Action

public ActionForward execute(ActionMapping mapping, ActionForm  form,
HttpServletRequest request, HttpServletResponse response) throws Exception
{
      NewServerActionForm nsaf = (NewServerActionForm) form;
      Integer userId =(Integer)request.getSession().getAttribute("userId");
      HashMap servers =
            HashMap)request.getSession().getAttribute("servers");
      Server newServer = new Server(nsaf.getServerName(),
            saf.getPop3Host(), nsaf.getPop3Port(), nsaf.getSmtpHost(),
            nsaf.getSmtpPort(), nsaf.getUsername(), nsaf.getPassword());
      servers.put(nsaf.getServerName(), newServer);
      DatabaseOperations.insertServer(userId.intValue(), newServer);
      return mapping.findForward(SUCCESS);
}
```

Ez az osztály az új szerveradatok felvételével kapcsolatos műveletekért felelős. Miután lekérdeztem a session változókat (userId, servers), felveszem az új adatokat az adatbázisba és a servers session változóba.

```
public class DeleteServerAction extends Action

public ActionForward execute(ActionMapping mapping, ActionForm  form,
HttpServletRequest request, HttpServletResponse response) throws Exception
{
      SelectActionForm saf = (SelectActionForm)form;
      HashMap servers =
(HashMap)request.getSession().getAttribute("servers");
      servers.remove(saf.getSelected());
      DatabaseOperations.deleteServer(saf.getSelected());
      return mapping.findForward(SUCCESS);
}
```

Ez az osztály egy legördülő listából kiválasztott szerver törléséért felelős. A kiválasztott elemet először a servers session változóból, majd az adatbázisból is törlöm.

```
public class SwitchServerAction extends Action

public ActionForward execute(ActionMapping mapping, ActionForm  form,
HttpServletRequest request, HttpServletResponse response) throws Exception
{
try {
      SelectActionForm saf = (SelectActionForm)form;
      HashMap servers = (HashMap)
request.getSession().getAttribute("servers");
      Server server = (Server)servers.get(saf.getSelected());
      Properties property = System.getProperties();;
      Session mSession = Session.getDefaultInstance(property);;
      Store store = mSession.getStore("pop3");;

      store.connect(server.getPop3Name(),
Integer.parseInt(server.getPop3Port()), server.getServerUsername(),
server.getServerPassword());

      Folder folder = store.getFolder("inbox");
      request.getSession().setAttribute("folder", folder);
}
```

```
catch (AuthenticationFailedException afe) {
      ActionMessages a = new ActionMessages();
      a.add("login.error", new ActionMessage("login.error"));
      saveErrors(request, a);
}
catch (Exception e) {
      e.printStackTrace();
      ActionMessages a = new ActionMessages();
      a.add("unable.to.connect", new ActionMessage("unable.to.connect"));
      saveErrors(request, a);
}
return mapping.findForward(SUCCESS);
}
```

Bejelentkezés után, ha van már szerver felvéve, és rákattintok a Read gombra, akkor ezt az eseményt ez az osztály fogja lekezelni. Tehát a legördülő listából kiválasztott szerverre fogok csatlakozni, és lekérem az ’inbox’ nevű mappáját, ami a beérkező leveleket hivatott tárolni. Ezt az inbox mappát fogom eltárolni egy folder nevű session változóba, melyből egy erre kifejlesztett JSP lap fogja kilistázni az e-maileket. Amennyiben a csatlakozás sikertelen, a kivételkezelő lép életbe, és hibaüzenettel tudatom a felhasználóval a hiba okát.

```
public class ReadAction extends Action

public ActionForward execute(ActionMapping mapping, ActionForm  form,
HttpServletRequest request, HttpServletResponse response) throws Exception
{
      Folder folder = (Folder)request.getSession().getAttribute("folder");
      int i = Integer.parseInt((String)request.getParameter("index"));
      try {
            folder.open(folder.READ_ONLY);
            ExtractMessage em = new ExtractMessage(folder.getMessage(i+1));
            request.getSession().setAttribute("ExtractMessage", em);
            folder.close(false);
      }
      catch(Exception e) {
            e.printStackTrace();
      }
      return mapping.findForward(SUCCESS);
}
```

Miután a JSP oldal kilistázta a bejövő e-maileket, ahhoz, hogy egy e-mailt elolvassunk rá kell kattintani egy listaelemre. Ennek a kattintásnak az eseménykezelője a fenti osztály. Lekérem a session változókból az előző osztályban elmentett folder változót, és egy index nevű változót, mely azt mondja meg, hogy a listában hányadik elemre kattintottam. Ezáltal egyértelműen beazonosítom, hogy mely e-mailre történt a kattintás, és ezt dolgozom fel úgy, hogy megjeleníthető legyen. A feldolgozást az ExtractMessage osztály végzi, melyet fentebb már bemutattam. Az újonnan létrehozott ExtractMessage objektumot elmentjük egy session változóba, az e-mail megjelenítő lap ennek az objektumnak a segítségével fogja megjeleníteni az email szöveges tartalmát, illetve csatolmányait.

```
public class DownloadAction extends Action

public ActionForward execute(ActionMapping mapping, ActionForm  form,
HttpServletRequest request, HttpServletResponse response) throws Exception
{
      ExtractMessage em =
(ExtractMessage)request.getSession().getAttribute("ExtractMessage");
      int i = Integer.parseInt((String)request.getParameter("index"));
      Attachment a = (Attachment)em.getAttachments().get(i);
      request.getSession().setAttribute("Attachment", a);
return mapping.findForward(SUCCESS);
}
```

Az üdvözlőlap alján lévő sáv arra szolgál, hogy a leendő csatolmányokat le lehessen tölteni. Ezeknek a csatolmányoknak a neve fog megjelenni link formájában, és ha valamelyikre rákattintok, akkor a fenti osztály execute metódusa fog lefutni. Először lekérdezem az előző osztályban beállított ExtractMessage session változót, és egy indexet, mely azt hivatott jelezni, hogy mely csatolmány nevére kattintottam. Ezek alapján egyszerűen ki lehet keresni, mely csatolmányra voltam kíváncsi. Az így megkapott Attachment típusú objektumot ugyancsak eltárolom egy Attachment nevű session változóba.

```
public class DeleteEmailAction extends Action

public ActionForward execute(ActionMapping mapping, ActionForm  form,
HttpServletRequest request, HttpServletResponse response) throws Exception
{
try {
      DeleteEmailActionForm daf = (DeleteEmailActionForm)form;
      Folder folder = (Folder) request.getSession().getAttribute("folder");
```

```
      String[] list = daf.getDeleted();
      if (list == null) return mapping.findForward(SUCCESS);
      folder.open(Folder.READ_WRITE);
      Message[] messages = folder.getMessages();
      for (int i = 0; i < list.length; ++i) {
            int index = Integer.parseInt(list[i]);
            messages[index].setFlag(Flags.Flag.DELETED, true);
      }
      folder.close(true);
}
catch (Exception e) {
      e.printStackTrace();
}
return mapping.findForward(SUCCESS);
}
```

A `DeleteActionForm` bean által szolgáltatott lista alapján meghatározom mely üzeneteket kell törölni. Ezt úgy tudom megtenni, hogy az írás-olvasásra megnyitott 'inbox' mappából a törölni kívánt `message` objektumok jelzőbitjét törlésre állítom be. A `folder.close(true)` művelettel véglegesítem a tranzakciót az email szerver felé.

```
public class SendEmailAction extends Action

private Server server;

public ActionForward execute(ActionMapping mapping, ActionForm  form,
HttpServletRequest request, HttpServletResponse response) throws Exception
{
try {
      SendEmailActionForm seaf = (SendEmailActionForm)form;
      HashMap servers =
(HashMap)request.getSession().getAttribute("servers");
      server = (Server)servers.get(seaf.getSelected());
      Properties property = System.getProperties();
      property.put("mail.smtp.host", server.getSmtpName());
      property.put("mail.smtp.port", server.getSmtpPort());
      property.put("mail.smtp.auth", "true");
      Session session = Session.getInstance(property, new Authenticator() {
      public PasswordAuthentication getPasswordAuthentication() {
```

```
return new
PasswordAuthentication(SendEmailAction.this.server.getServerUsername(),
SendEmailAction.this.server.getServerPassword());
}});
      MimeMessage mail = new MimeMessage(session);
      InternetAddress[] recipients = new InternetAddress[1];
      recipients[0] = new InternetAddress(seaf.getTo());
      mail.setRecipients(Message.RecipientType.TO, recipients);
      mail.setSubject(seaf.getSubject());
      mail.setContent(seaf.getText(), "text/plain");
      Transport.send(mail);
}
catch (Exception e) {
      e.printStackTrace();
}
return mapping.findForward(SUCCESS);
}
```

Amint a nevéből is lehet következtetni, ez az osztály egy e-mailt fog postázni. Feladatom csupán annyi volt, hogy a session változókból lekérdezzem, hogy melyik szerver segítségét kérte a postázásra a felhasználó, beállítsam a környezeti változókat, illetve beállítsak egy újonnan létrehozott MimeMessage objektum attribútumait. Az elküldésért a Transaction osztály send statikus metódusa felelős.

Jelen pillanatban csak ’text/plan’ tipusú emaileket lehet küldeni csatolmányok nélkül, azonban a közeljövőben tervezem ’text/html’ típusú e-mailek küldésének megvalósítását csatolmányok hozzáadásával.

4.3 Rendszer hatékonysága

A rendszer legnagyobb hátránya a lassúsága, ugyanis a különböző e-mail szerverekről az e-mailek letöltése időigényes, és mivel a rendszer webes felületű, nincs lehetőség arra, hogy a már egyszer letöltött e-mailt, legközelebb ne töltsük le újra. Csakhogy viszonyítási pontot nyújtsak, a freemail.hu-ról kb. 130 csatolmány nélküli email letöltése kb. 20mp-be kerül. Ennek tudatában igyekeztem a lehető leghatékonyabban leprogramozni a feladatot, kevés osztályt, és objektumot használni.

A rendszer korábbi verziója még ennél is lassabb volt, mivel az e-mailek kezelésére kidolgoztam egy komplex adatszerkezetet a megfelelő saját készítésű osztályaival együtt, ez

azonban ha nem is annyira észrevehetően, de ugyancsak lassították a futási időt. Ezért újraírtam az e-mailt feldolgozó modult kicsit javítva így a futási időn.

Meggyőződésem, hogy ennél már nagyságrendekkel nem lehet javítani a futási időn, mivel az e-mail-ek letöltése egy teljesen a Sun által megírt algoritmus alapján történik.

5. Összefoglalás

A WebMail fejlesztése során számos problémába ütköztem, melyeknek megoldásával hasznos, életszerű tapasztalatra tettem szert. Egy konkrét alkalmazás fejlesztése révén ismerkedtem meg a napjainkban használt technológiákkal, eszközökkel, melyeknek ismerete mára már elengedhetetlen az informatikai munkaerőpiacon.

Az alkalmazással kapcsolatban rengeteg elképzelés vár megvalósításra, ezek közül csak néhányat sorolnék fel:

- ’text/html’alapú emailek támogatása
- csatolmányok küldése
- lehetőség szerint csak az olvasatlan e-mailek letöltése
- ’inbox’-on kívüli más mappák kezelése

6. Irodalomjegyzék

- Nyékyné Gaizler Judit: Java 2 , Útikalauz programozóknak 1.3, ELTE , Budapest, 2001
- Nyékyné Gaizler Judit: J2EE, útikalauz Java programozóknak, ELTE TTK Hallgatói alapítvány, Budapest, 2002
- Vég Csaba: Alkalmazásfejlesztés az Unified Modeling Language szabványos jelöléseivel, Logos 2000, Debrecen, 1999
- Java dokumentáció: http://java.sun.com/javase/6/docs/api
- Javascript dokumentáció: http://www.w3schools.com/js/
- Struts dokumentáció: http://struts.apache.org/

7. Függelék

Struts konfigurációs fájljának egy részlete a WebMail alkalmazáshoz:

```
<struts-config>
      <form-beans>
      <form-bean name="SendEmailActionForm"
type="struts.bean.SendEmailActionForm"/>
      <form-bean name="SelectActionForm"
type="struts.bean.SelectActionForm"/>
      <form-bean name="NewServerActionForm"
type="struts.bean.NewServerActionForm"/>
      <form-bean name="RegistrationActionForm"
type="struts.bean.RegistrationActionForm"/>
      <form-bean name="LoginActionForm"
type="struts.bean.LoginActionForm"/>
      <form-bean name="DeleteEmailActionForm"
type="struts.bean.DeleteEmailActionForm"/>
      </form-beans>
…
      <action-mappings>
            <action input="/login.jsp"
                  name="LoginActionForm"
                  path="/login"
                  scope="request"
                  type="struts.action.LoginAction"
                  validate="false">
            <forward name="failure" path="/login.jsp" />
            <forward name="success" path="/readmail.jsp" />
            </action>

            <action path="/read"
                  scope="request"
                  type="struts.action.ReadAction">
            <forward name="success" path="/read.jsp" />
            </action>
```

```
<action name="DeleteEmailActionForm"
      scope="request"
      path="/delete"
      type="struts.action.DeleteEmailAction">
      <forward name="success" path="/inbox.jsp" />
</action>

<action input="/registration.jsp"
      name="RegistrationActionForm"
      scope="request"
      path="/registration"
      type="struts.action.RegistrationAction"
      validation="false">
<forward name="failure" path="/registration.jsp"/>
<forward name="success" path="/login.jsp"/>
</action>

<action input="/functions.jsp"
      name="SelectActionForm"
      scope="request"
      path="/switch"
      type="struts.action.SwitchAction"
      validate="false">
<forward name="success" path="/inbox.jsp"/>
</action>

<action input="/new_server.jsp"
      name="NewServerActionForm"
      scope="request"
      path="/new_server"
      type="struts.action.NewSeverAction"
      validate="false">
<forward name="success" path="/functions.jsp"/>
</action>

<action input="/delete_server.jsp"
```

```
            name="SelectActionForm"
            path="/delete_server"
            scope="session"
            type="struts.action.DeleteServerAction"
            validate="false">
        <forward name="success" path="/functions.jsp"/>
        </action>

        <action input="/write.jsp"
            name="SendEmailActionForm"
            path="/send"
            scope="session"
            type="struts.action.SendEmailAction"
            validate="false">
        <forward name="success" path="/functions.jsp"/>
        </action>

        <action path="/download"
            type="struts.action.DownloadAction">
        <forward name="success" path="/download.jsp"/>
        </action>
    </action-mappings>
    …
</struts-config>
```